EJB™ Development Using Borland® JBuilder® 8 and BEA WebLogic Server™ 7.0

Jumpstart development, deployment, optimization, and debugging EJB™

A Borland White Paper

by Sudhansu Pati, Systems Engineer

December 2002

Contents

Introduction 4

Working with session beans 5

Create a project 5
Make WebLogic® the target server 5
Create an EJB™ module 5
Visually create a session bean 6
Deploy and run session bean 8
Create a runtime configuration 8
Make the project 10
Run WebLogic Server™ within JBuilder® 11
Deploy JAR file for EJB™ to WebLogic® 11
Remote deployment 14
Test the session bean 15

Optimizing EJB™ applications 19

Optimizeit™ Code Coverage 19
Optimizeit™ Profiler 19
Optimizeit™ Thread Debugger 19

Working with entity beans 23

Create an entity bean 23
Import a database schema 24
Generate an entity bean 25
Create a session bean 26
Assign entity bean reference to session bean 28
Create data source entries in WebLogic 29
Create a connection pool in WebLogic 30
Create a transactional data source 33
Test the data source 34

Deploy and run EJBs in WebLogic 34
Test the entity bean 35
Run the test client 36

Remote debugging of EJBs 36

Provide remote debug options 37
Create runtime configuration for remote debugging 37
Assign breakpoints 40
Run WebLogic Server in debug mode outside of JBuilder 40
Attach JBuilder to WebLogic remotely 41
Run the test client to debug the EJBs 42

Working with Message Driven Beans 43

Create JMS Configurations in WebLogic® 44

Create a Connection Factory 44
Create a JMS Store 45
Create a JMS Server 46
Create a Queue 47
Create a Message Driven Bean 48
Deploy and run the MDB 50
Test the MDB 50

Troubleshooting 53

Useful links 62

Feedback and suggestions 63

Introduction

Borland® JBuilder® Enterprise is a comprehensive set of award-winning visual development tools for creating enterprise-scale applications written entirely in the Java™ programming language for the Java 2 platform.

JBuilder 8 Enterprise provides out-of-the-box integration with BEA WebLogic Server™ 5.1, 6.x , and 7.0.

This paper provides an overview of creating, testing , debugging, optimizing, and deploying Enterprise JavaBeans™ (EJB™) using Borland JBuilder 8 Enterprise and BEA WebLogic Server 7.0 (SP1).

The other platforms and software used to create the exercises are Optimizeit™ Suite 5.0, Windows XP™ and Microsoft® Internet Explorer 6.0.

WebLogic Server 7.x needs to be installed and configured with JBuilder 8 to work with the exercises described in this paper. Refer to the paper "JBuilder 8 Configuration with J2EE Application Servers" for installation and configuration instructions for JBuilder 8 Enterprise with WebLogic Server 7.0 (SP1).

The audiences for this paper are Project Managers, architects, and developers. The audience requires basic knowledge of Java and EJB to fully understand the contents of this paper.

Working with session beans

For this exercise, let's create a simple session bean using JBuilder wizards, deploy, and run it in WebLogic, and optimize it using Optimizeit Suite.

Create a project

From JBuilder main menubar, click **File -> New -> Project** tab from **Object Gallery** -> click **Project** -> click **OK**.

In the Project Wizard, enter the **Project Name** as **TestSesProject** and the **Directory Name** as **E:\Demo\WebLogic\TestSesProject**.

Make WebLogic® the target server

From JBuilder main menubar, click **Project -> Project Properties** -> click **Server** tab. Select **WebLogic Application Server 7.x** from the combo box. The radio button Single Server for all services in project should already be the default selection. Click **OK**.

Create an EJB™ module

From the JBuilder menubar, click **File -> New -> Enterprise** tab of **Object Gallery** -> **EJB Module.** Hit **OK**. In the EJB Module Wizard, enter the module name as **TestSesMod** and select the EJB version as **EJB 2.0 compliant**.

See Figure 1 for the EJB Module Wizard.

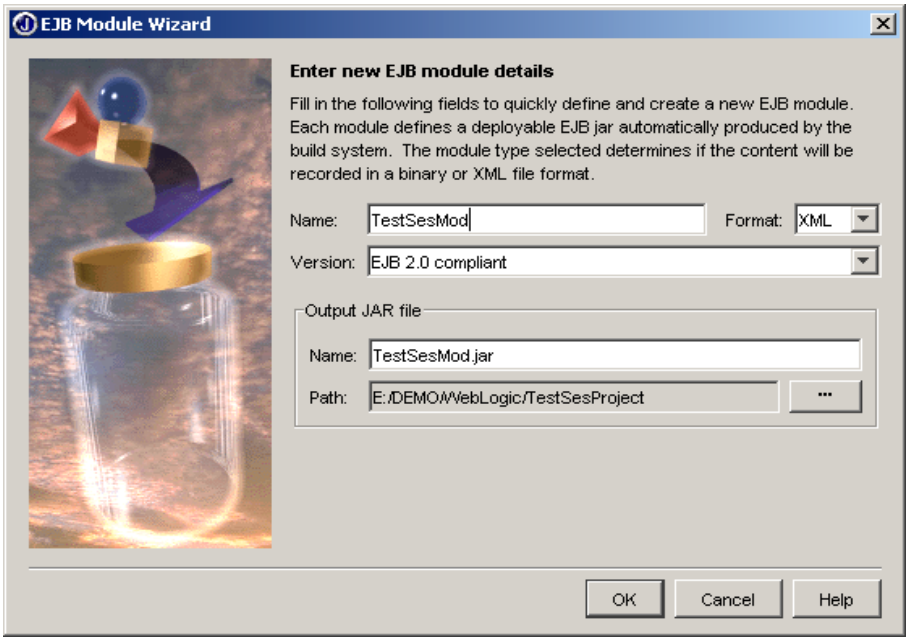
EJB Module Wizard
Enter new EJB module details
Fill in the following fields to quickly define and create a new EJB module. Each module defines a deployable EJB jar automatically produced by the build system. The module type selected determines if the content will be recorded in a binary or XML file format.
Name: TestSesMod
Format: XML
Version: EJB 2.0 compliant
Output JAR file
Name: TestSesMod.jar
Path: E:/DEMO/WebLogic/TestSesProject
OK
Cancel
Help

***Figure 1**: JBuilder EJB Module Wizard to create a visual EJB hosting environment*

Visually create a session bean

With JBuilder, EJBs can be created visually. Double-click **EJBModule TestSesMod**, right-click the editor pane. Click **Create EJB**. Click **Session Bean**. Enter the Bean name as **TestSes** in the **Bean Properties** window.

> **Note**:The previous step automatically creates three EJB files (TestSesBean.java, TestSes.java, and TestSesHome.java). You can see these files in JBuilder project tree.

To change the package name, click **Classes and packages** button. Enter the package name as **com.borland.demo.ejb**

See Figure 2 for details.

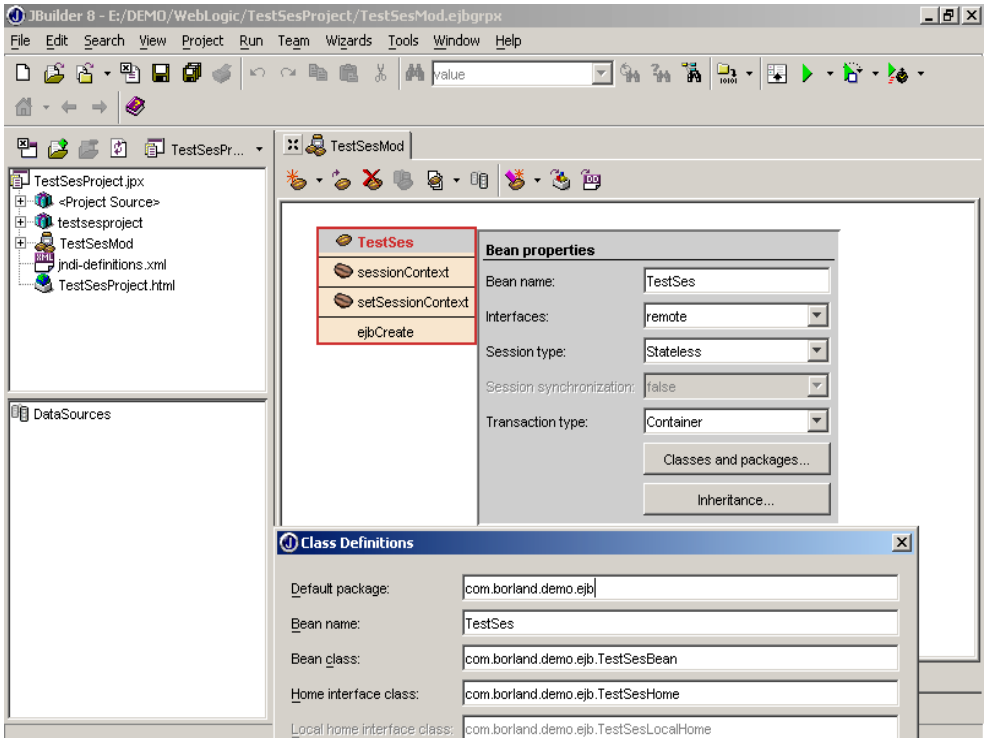

***Figure 2**: Creating a session bean visually using JBuilder visual EJB designer*

To add a method in the session bean, right-click **TestSes**. Click **Add**. Click **Method**. Enter the method name as **getMessage()**, return type as **string** and interfaces as **remote**.

Right-click the **EJB TestSes** and click **View Bean Source**. This opens up the TestSesBean.java file in the JBuilder editor pane.

Change the method getMessage() as follows.

```
public String getMessage() {
    return "Testing Successful";

}
```

Deploy and run session bean

Create a runtime configuration

JBuilder allows users to create runtime configurations for running different applications.

> **Note**: A runtime configuration is a configuration to run applications in JBuilder 8. Runtime configuration is essential when your project has multiple runnable files or applications, and you want to run a specific application.

From the JBuilder main menubar, click **Run -> Configurations**. The Project Properties window should show up with Run tab selected. Click **New** to create a new runtime configuration.

In the Runtime Configuration Properties window, enter the following values.

Name:	`Start WebLogic 7.0 [Users can enter any name here]`
Build target:	`<None>`
Type:	`Server`

See Figure 3 for details.

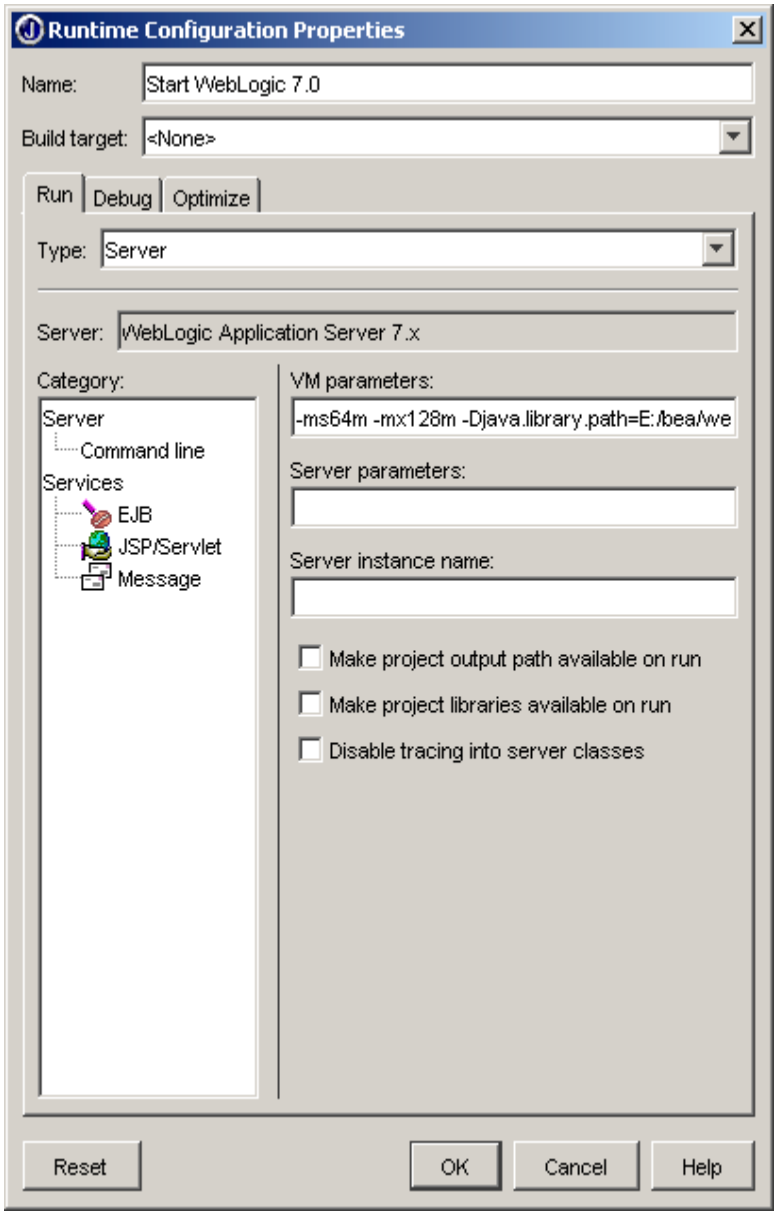

***Figure 3**: Runtime configuration to run WebLogic within JBuilder*

Click **OK** and now you can see a runtime configuration called **Start WebLogic 7.0** in Project Properties window. Click **OK**.

Make the project

Make the project (**Ctrl +Shift + F9**).

> **Note**: After the Make, the WebLogic specific container generated files and deployment descriptors are created in JBuilder. Expand TestSesBean.java to view the container-generated files and double-click TestSesMod.jar to view the deployment descriptors in the structure pane.

See Figure 4 for details.

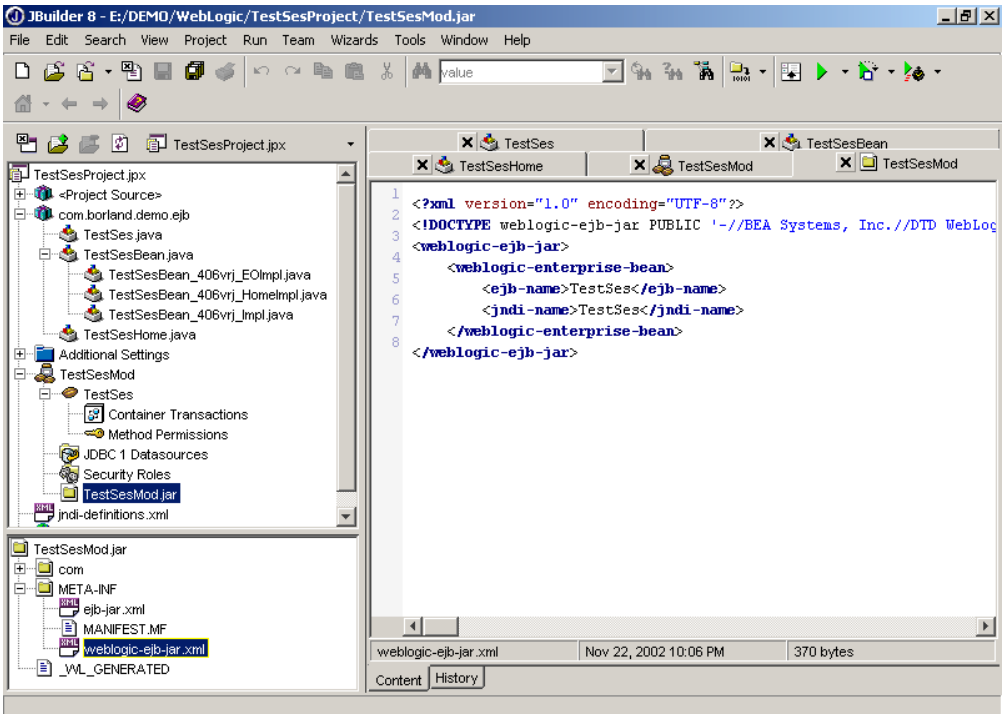

***Figure 4**: JBuilder has created the container generated files and WebLogic deployment descriptors*

> **Note**: If you like to save the deployment descriptors to the disk/network, right-click the **EJBModule TestSesMod** and click **Properties**. Click **Build** tab, then **EJB** tab, and click the checkbox against **Copy descriptors** and provide the Directory.

Run WebLogic Server™ within JBuilder®

Click on the **Run Project** icon from the JBuilder toolbar and click **Start WebLogic 7.0.** Now, WebLogic Server 7.0 will run within JBuilder.

See Figure 5 for details.

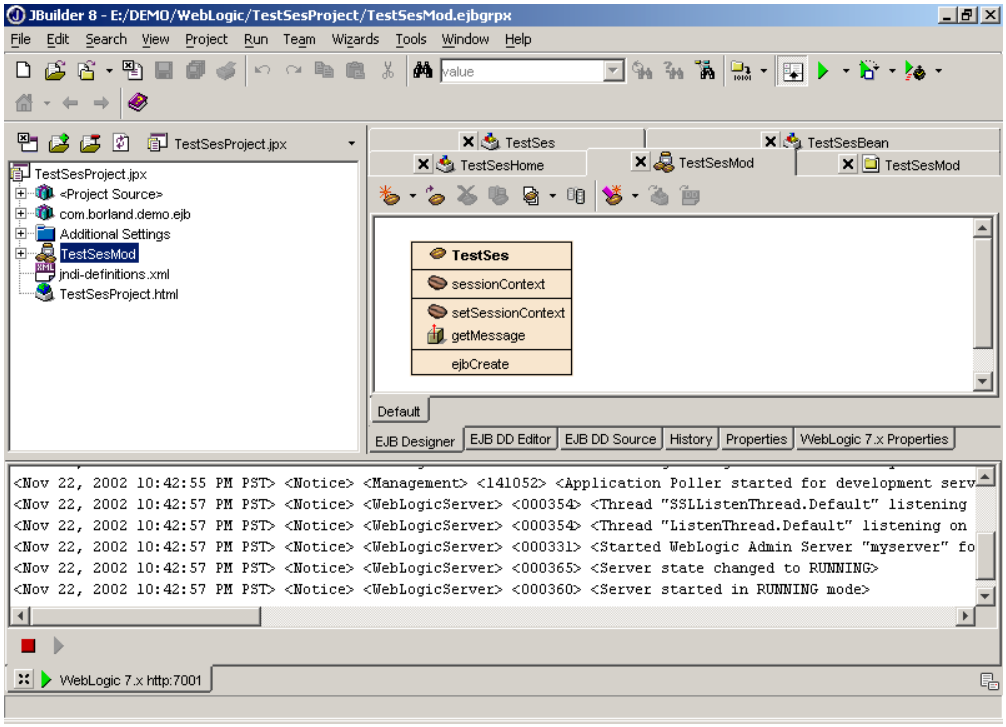

***Figure 5**: WebLogic Server 7.0 running within JBuilder 8*

Deploy JAR file for EJB™ to WebLogic®

Right-click **TestSesMod.jar**. Click **Deploy options for TestSesMod.jar**. Click **Deploy**.

JBuilder starts another tab called **WebLogic Application Server 7.x Enterprise Deployer** in the message window to show the deployment status.

See Figure 6 for details.

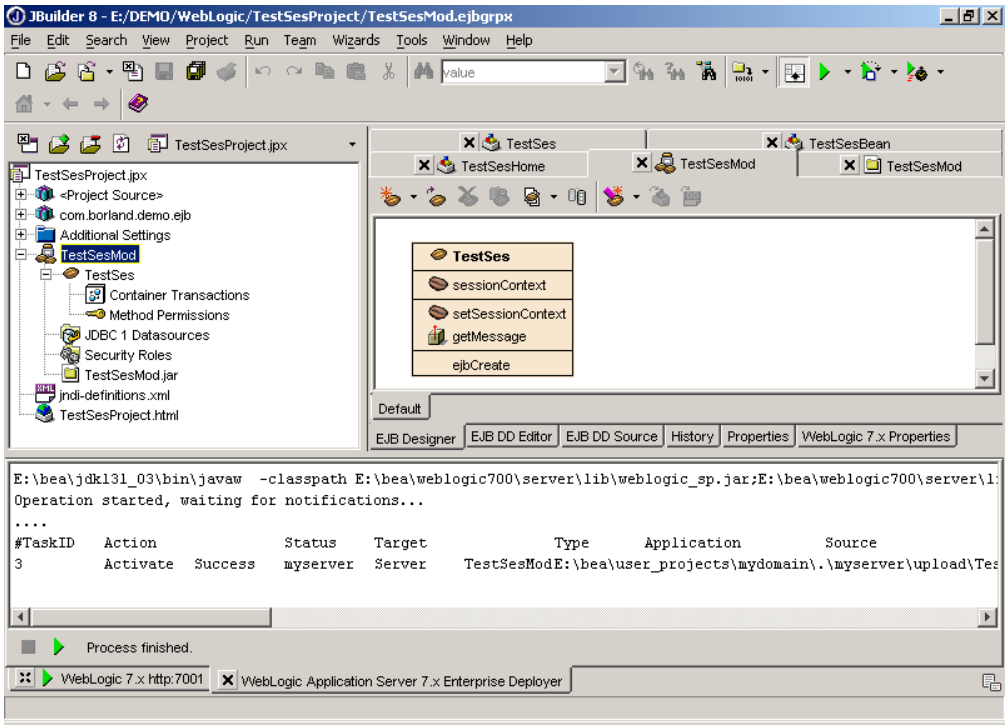

***Figure 6**: JBuilder displaying the deployment status of JAR file in a separate tab*

> **Note**: JBuilder provides **hot-deployment** to WebLogic. The WebLogic Server need not be restarted upon deploying any archive to WebLogic. Also, JBuilder provides **Redeploy** and **Undeploy** features.

To further check the deployment within Weblogic Server, start WebLogic Server Console. To start WebLogic Server Console, from JBuilder main menubar, click **Tools -> WebLogic 7.x Admin Console**. Now the WebLogic Server console is opened in your favorite Internet browser.

From WebLogic Server Console, expand **mydomain**, expand **Deployments,** and click **EJB**. The deployed module TestSesMod should appear in the right-hand side pane.

See Figure 7 for WebLogic Console and deployed TestSes within WebLogic.

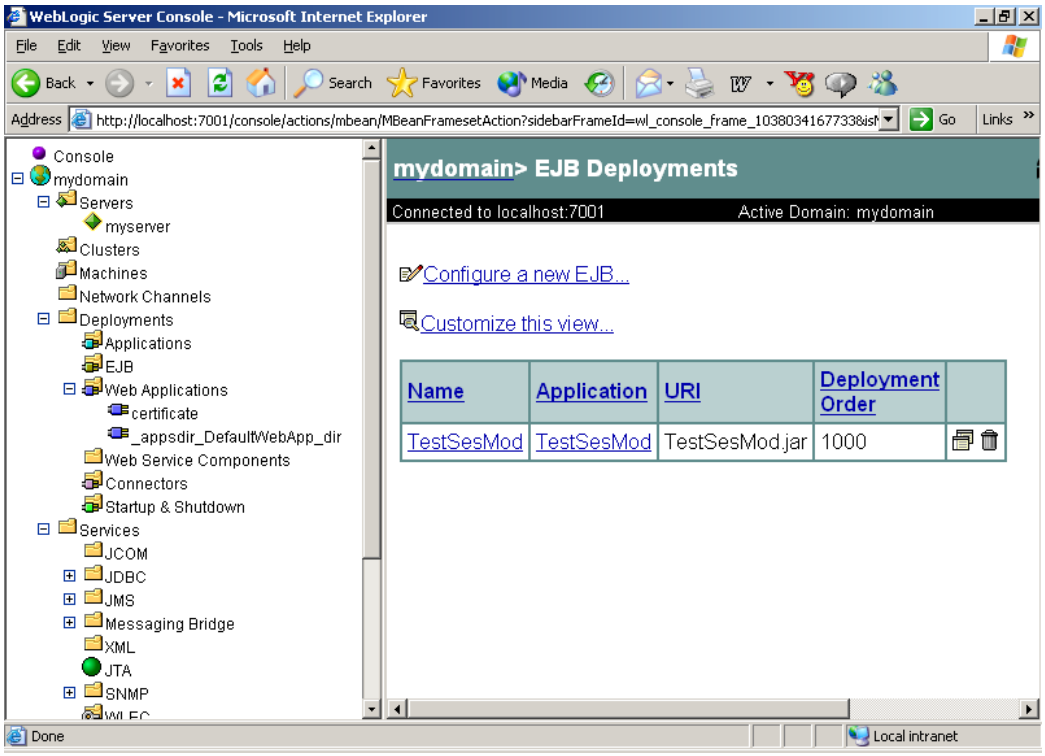

***Figure** 7: WebLogic Server Console showing deployed module TestSesMod*

Note: If you don't see WebLogic 7.x Admin Console under JBuilder 8 Tools menu, it is because you haven't selected this option during server configuration. Click **Tools -> Configure Servers** -> Select **WebLogic Application Server 7.x** from left-hand panel of Configure Servers window, click **Custom** tab from the right-hand panel, and make sure that checkbox against Add a Admin Console item to the Tools menu is checked. You have to also provide the Web browser path, which is the path to the executable for your favorite browser (i.e. C:/Program Files/Internet Explorer/IEXPLORER.EXE).

Remote deployment

This section is an optional step and provides tips on how to deploy an archive to a remote WebLogic Server.

> **Note**: For deployment of archives (JAR, WAR, and EAR) to a remote WebLogic Server, a local installation of the application server is needed because the deployment tool needs local WebLogic utilities for remote deployment.

From JBuilder main menubar, click **Tools -> Enterprise Deployment**. A window called **Weblogic 7.x Deploy Settings** pops up. Modify the Admin URL field to point to the remote server with the remote WebLogic Server's host name and port number (http://<servername>:<port number>).

Refer to Figure 8 for details on remote deployment options.

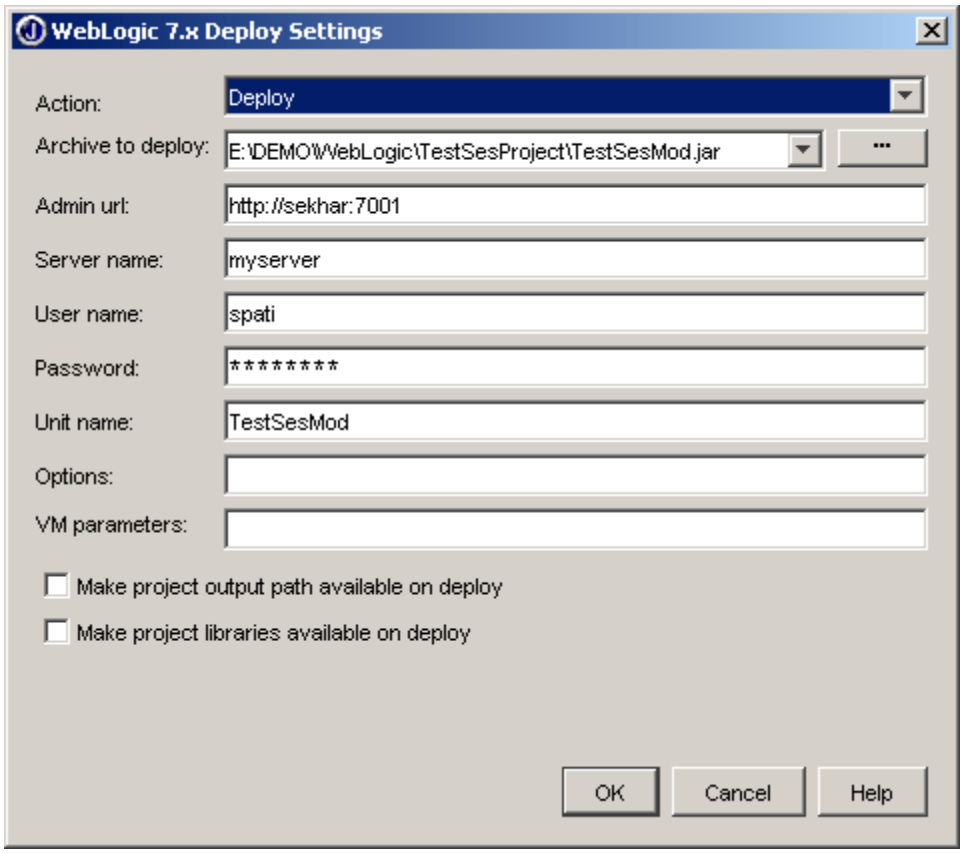

***Figure 8**: WebLogic 7.x Deploy Settings window for deploying to Remote WebLogic Server*

Test the session bean

This exercise explains how to test the session bean using a Java client. JBuilder provides a wizard to create a test client to test the EJBs.

Create a test client

From the JBuilder menubar, click **File -> New -> Enterprise** tab of **Object Gallery -> EJB Test Client**. Click **OK**.

The EJB Test Client Wizard pops up. In EJB Test Client Wizard–Step 1 of 3, make sure that the radio button against **Application** is checked. Click **Next**.

See Figure 9 for details.

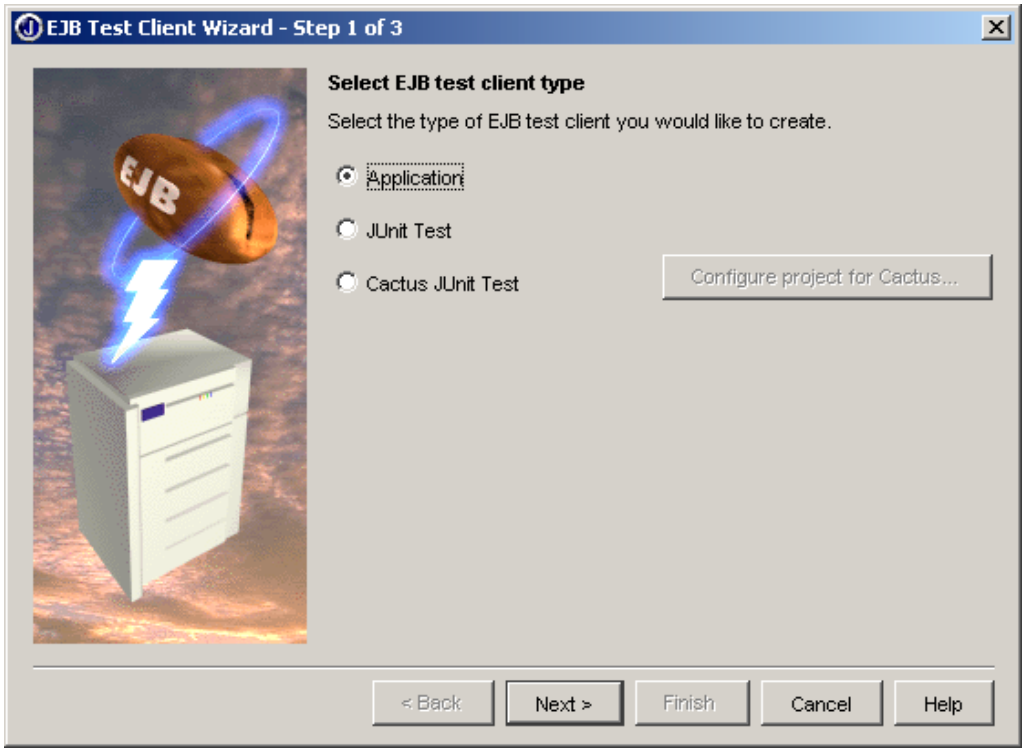

***Figure 9**: EJB Test Client Wizard: creating a test client for session bean TestSes*

In EJB Test Client Wizard–Step 2 of 3, leave the EJB Name as TestSes, enter the package name as **com.borland.demo.ejb** and Class name as **TestSesTestClient**. Click **Finish**.

> **Note**: JBuilder can create a test client for an application, JUnit Test, or Cactus JUnit Test. When a Test Client is created for an EJB, JBuilder automatically wraps all the business methods in the Test Client and makes it easier for the developers to use without having to write any manual code.

JBuilder has created a runtime configuration TestSesTestClient to run the test client.

Expand the package **com.borland.demo.ejb** and change the main() method of **TestSesTestClient.java** as follows.

```
public static void main(String[] args) {
    TestSesTestClient client = new TestSesTestClient();
    try {
      client.create();
      String message = client.getMessage();
      System.out.println("Message from the Test Client = " + message);
    }
catch (Exception ex) {
    }
}
```

Make the TestSesTestClient.java (right-click **TestSesTestClient.java** and click **Make**) to make sure that there are no compilation errors.

Run the test client

To run the test client, right-click **TestSesTestClient.java**, click **Run using TestSesTestClient**. Alternatively, you can just click the R**un** icon and click **TestSesTestClient** to run the test client.

See Figure 10 shows the test client running within JBuilder.

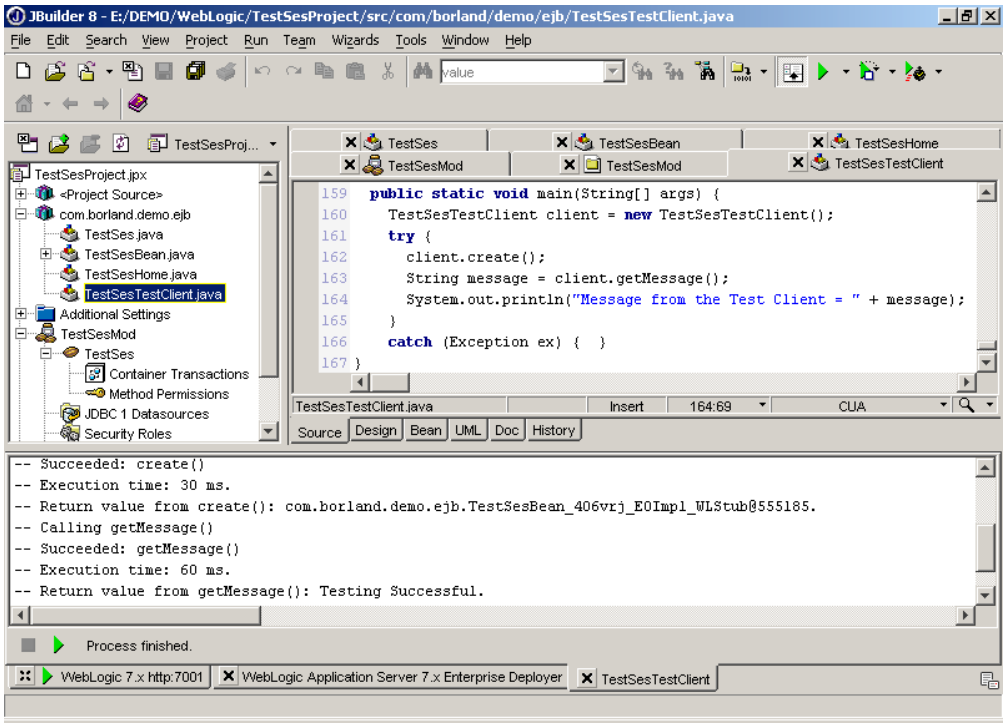

***Figure 10**: Test Client running within JBuilder and showing outputs from EJB method getMessage()*

The test client runs within JBuilder by creating another tab in the message pane. You can see the successful execution of the Test Client from the last line `Message From the Test Client = Testing Successful.`

Optimizing EJB™ applications

JBuilder 8 Enterprises has out-of-the-box integration with Optimizeit Suite.

Optimizeit Suite is a separate installation, available for purchase or as a trial download from the Borland Web site. Please visit the following link for downloading a trial editon of Optimizeit Suite 5.0:

http://www.borland.com/products/downloads/download_optimizeit.html

Optimizeit Suite has three different tools (Optimizeit™ Code Coverage, Optimizeit™ Profiler, and Optimizeit™ Thread Debugger). For more information, visit the link http://www.borland.com/optimizeit/index.html

Note: The Code Coverage tool comes with JBuilder 8 Enterprise. So a separate installation of Optimizeit Suite is not required for using the Code Coverage tool.

Optimizeit™ Code Coverage

Optimizeit Code Coverage enables developers to view the execution frequency of each class, method, and line of code, and it helps to reduce application footprint by removing dead code.

Optimizeit™ Profiler

Optimizeit Profiler enables developers to maximize application speed and reliability and to quickly isolate critical code that requires performance improvements.

Optimizeit™ Thread Debugger

Optimizeit Thread Debugger helps developers to solve mysterious thread issues easily, get the status of all threads and monitors in real time, avoid thread starvation and contentions that lead to crashes, and predict deadlocks before they occur.

Note: During Optimizeit installation, the installer automatically detects JBuilder installation from user's machine and prompts for integrating with JBuilder. If the user says OK, then both JBuilder and Optimizeit are configured for tight integration.

Optimizeit Suite reads the JVMs and helps the developers in profiling, thread optimizing, and code covering their applications. If the JVM runs a Java application, Optimizeit can attach to the application and display the memory, thread, and other information in a user friendly way which helps the developers to optimize their applications.

Optimizeit Suite is a very flexible tool, and it can attach with various market-leading IDEs and application servers.

Visit the link http://info.borland.com/techpubs/optimizeit/ to find the supported J2EE application servers, documentation, and other information about Optimizeit Suite 5.

For this case, let's attach Optimizeit Suite with WebLogic Server 7.x. Click the **Optimize Project** icon from JBuilder toolbar and click **Start WebLogic 7.0**. The Runtime Configuration window pops up and allows users to select one of the three tools (Profiler, Thread Debugger, Code Coverage). See Figure 11 for details.

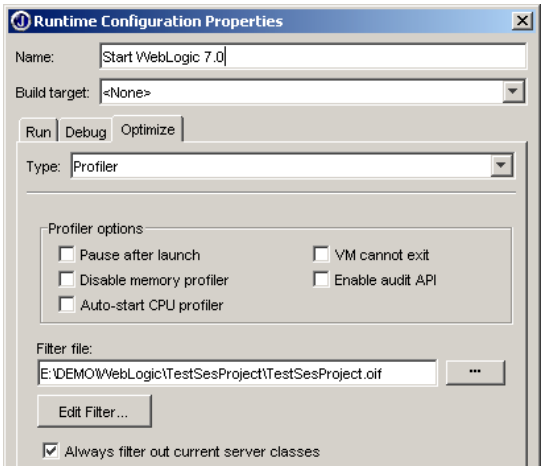

***Figure 11**: Runtime configuration window allows to choose which Optimizeit tool to use*

Select **Profiler** from the combo box and click **OK** twice. WebLogic 7.x starts within JBuilder 8. Note that Optimizet Profiler is attached to WebLogic 7.x, and Optimizeit Profiler starts profiling the real-time JVM information.

See Figure 12 for details.

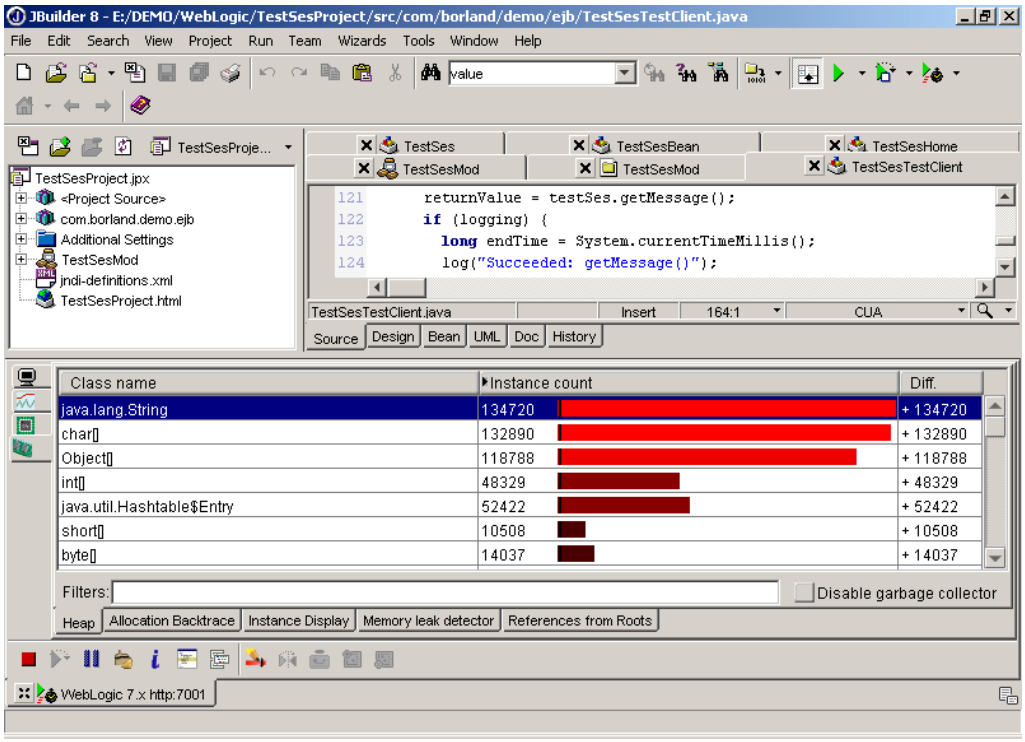

***Figure 12**: Optimizeit Profiler attached to WebLogic 7.x within JBuilder 8*

To go to normal WebLogic output window, click the **Console output, input, and errors** icon from the left. Also, you can click **Show Virtual Machine** information and **Show CPU Profiler** for different operations.

The Profiler shows information about all the classes available in the JVM. If you would rather see the information about your application only, type **com.borland.demo.ejb.TestSes*** at the Filters text area. This will show you the profiling information of the session bean TestSes only.

To attach Optimizeit Thread Debugger with WebLogic Server 7.x, click **Run -> Configurations** -> select **Start WebLogic 7.0** -> **Edit** -> click **Optimize** -> Select **Thread Debugger** from the Type combo box. See Figure 13 for Thread Debugger starting in JBuilder and attaching WebLogic Server 7.x.

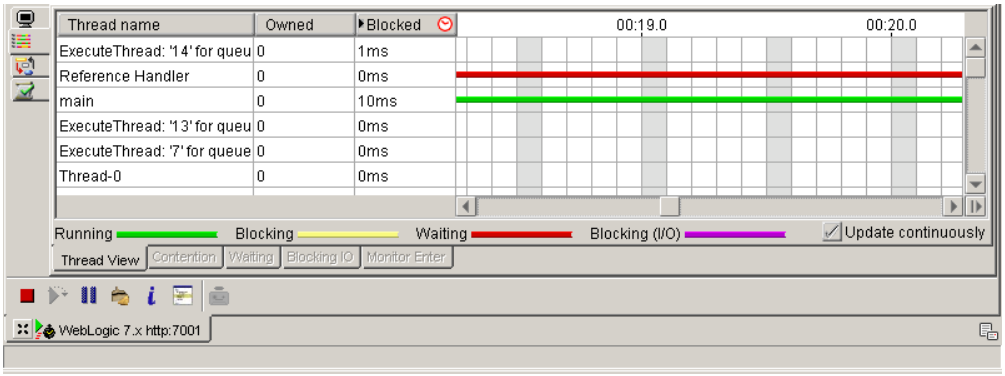

***Figure 13**: OptimizeIt Thread Debugger started in JBuilder 8 and attached with WebLogic Server 7.x*

Optimizeit Code Coverage can be started in the similar way. Type **com.borland.demo.ejb.TestSes*** at the Filters textbox. Now run the test client to see the behavior of the Code Coverage tool. The coverage jumps from 0 to 62.5%.

See Figure 14 for Optimizeit Code Coverage tool within JBuilder 8.

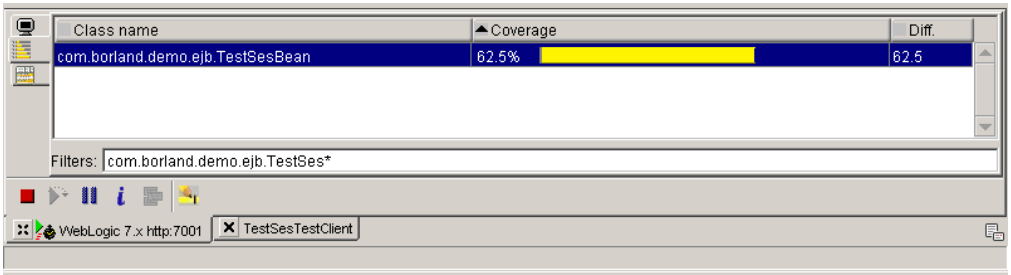

***Figure 14**: Optimizeit Code Coverage tool started in JBuilder 8 and attached with WebLogic Server 7.x*

There are lot of features available in Optimizeit Suite, which helps to detect critical memory leaks, performance issues, deadlocks etc. This paper just explained a snapshot of some of the features. Read the Optimizeit Suite product documentation for more information about other features of Optimizeit Suite.

Working with entity beans

Create a project, assign WebLogic Server 7.x as the target application server for the project, and create an EJB module. To get help on how to perform the above operations, refer to the previous chapter "Working with session beans."

For this exercise, enter the name of the project as **TestEntProject** in the directory **E:\Demo\WebLogic\TestEntProject** and enter the EJB module name as **TestEntMod**.

Create an entity bean

An entity bean represents the state of real-time data from the database table(s). Using JBuilder, users can create entity beans in following ways:

- First create entity beans and then generate a database from the entity beans.
- Generate entity bean from existing database.

For this exercise, lets focus on generating entity beans from an existing database.

> **Note**: To work with entity beans, WebLogic Server needs to have the database driver archive for the database in its classpath. From JBuilder main menubar, **click Tools -> Configure Servers**. From the left-hand side, select **WebLogic Application Server 7.x** and make sure that the database driver JAR file is added under the Class tab in the right-hand side.

Borland JDataStore™ database is already shipped with JBuilder 8. To add JDataStore driver archive, add **jdsserver.jar** from <**JBuilder8_Installed_Dir>\lib.**

Import a database schema

Right-click on the **editor pane**. Click **Import Schema from Database**. That brings up the Database Schema Provider window. Enter the **Database Driver, URL, Username, Password**, and **JNDI** name.

For this exercise, JDataStore will be used as the database. Enter the following values in the Database Schema Provider window:

Driver: `com.borland.datastore.jdbc.DataStoreDriver`

URL:
`jdbc:borland:dsremote://localhost//E:\Borland\JBuilder8\samples\JDataStore\datastores\employee.jds`

JNDI name: `JDSDataSource`

Username: `Sample`

Password: `<leave this empty>`

See Figure 15 for details.

> **Note**: JBuilder automatically creates a data source named JDSDataSource and displays all the tables under the data source. Check the structure pane of JBuilder for the data source and associated tables (automatically discovered by JBuilder).

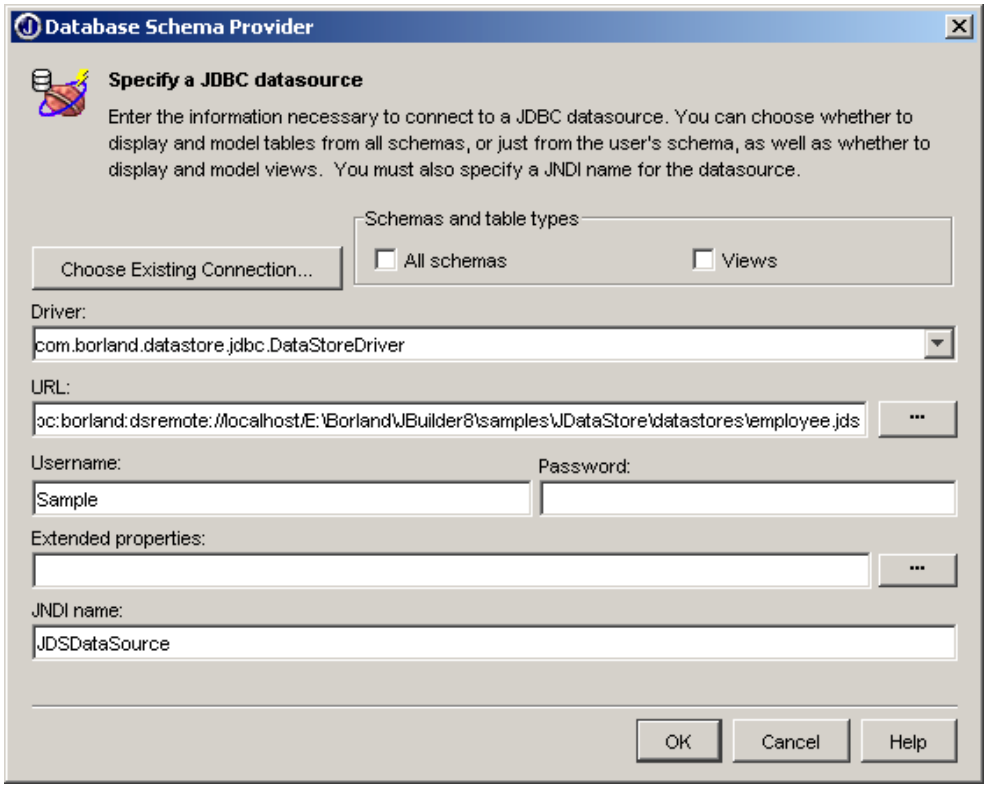
Database Schema Provider
Specify a JDBC datasource
Enter the information necessary to connect to a JDBC datasource. You can choose whether to display and model tables from all schemas, or just from the user's schema, as well as whether to display and model views. You must also specify a JNDI name for the datasource.
Schemas and table types
All schemas
Views
Choose Existing Connection...
Driver:
com.borland.datastore.jdbc.DataStoreDriver
URL:
pc:borland:dsremote://localhost/E:\Borland\JBuilder8\samples\JDataStore\datastores\employee.jds
Username:
Sample
Password:
Extended properties:
JNDI name:
JDSDataSource
OK
Cancel
Help

***Figure 15**: JBuilder Database Schema Provider—to create entity beans from database schema*

Generate an entity bean

Expand JDSDataSource from the JBuilder structure pane to see the list of tables available in the database. You can create entity beans visually just by right-clicking a table name and clicking Create CMP 2.0 Entity Bean or Create BMP Entity Bean.

For this exercise, let's create a CMP 2.0 Entity Bean for EMPLOYEE Table. Right-click **EMPLOYEE** from the structure pane and click **Create CMP 2.0 Entity Bean**. This creates a Visual Employee Entity Bean in the editor pane.

To change the package name, click the entity bean **Employee**, click **classes and packages** from the Bean Properties window and enter the package name as **com.borland.demo.ejb** in the Default package field of Class Definitions window. Click **OK**.

Note: JBuilder creates the EJB files Employee.java, EmployeeBean.java, and EmployeeHome.java in the package com.borland.demo.ejb. To check the source code, right-click the entity bean **Employee** and click **View bean source**.

Create a session bean

Please refer to the previous chapter "Working with session beans" for details on how to create a session bean.

For this exercise, enter the name of the session bean as **EmployeeSes** and the package name as **com.borland.demo.ejb**.

Add a method (right-click **EmployeeSes** in the editor pane, click **add**, click **method) to EmployeeSes** and provide the following parameters.

Method name:	`getEmpName`
Return type:	`String`
Input parameters:	`Short empNo`
Interfaces:	`remote`

See Figure 16 for details.

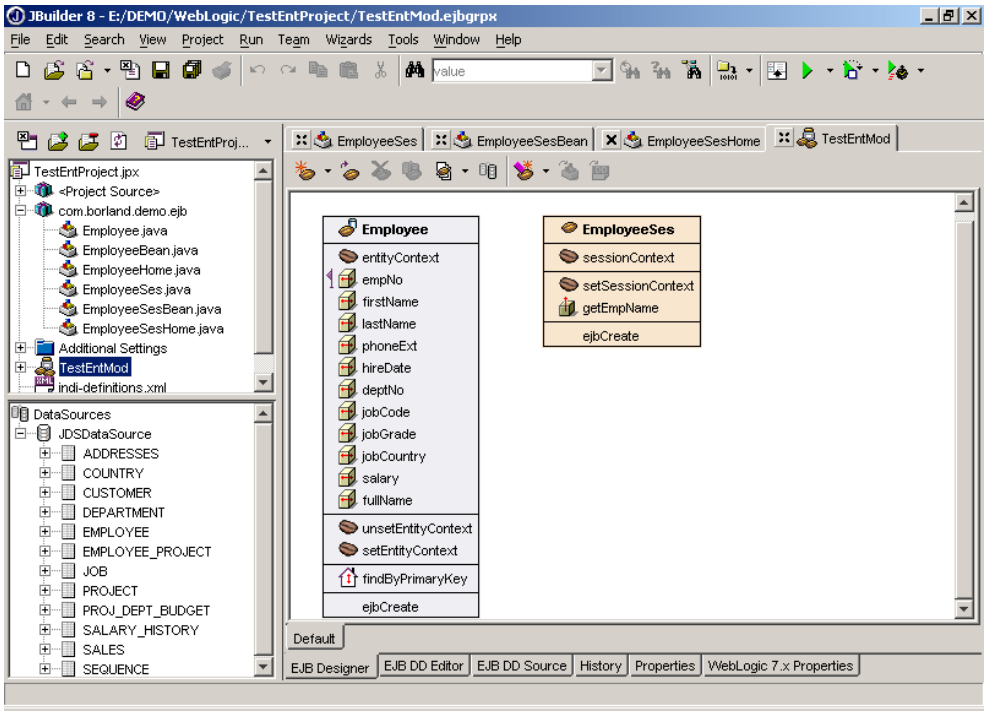

***Figure 16**: Visual EJBs in JBuilder displaying the fields and methods of entity and session beans*

Open **EmployeeSesBean.java** in the editor pane (right-click **EmployeeSes** and click **View bean source**) and change the **getEmpName(Short empNo)** method as follows"

```
public String getEmpName(Short empNo) {
    String empName = "name not found";
    try {
      javax.naming.Context context = new
javax.naming.InitialContext();
      Object object = context.lookup("java:comp/env/ejb/Employee");
      EmployeeHome empHome = (EmployeeHome)
javax.rmi.PortableRemoteObject.narrow(object, EmployeeHome.class);
      Employee emp = empHome.findByPrimaryKey(empNo);
      empName = emp.getFirstName() + " " + emp.getLastName();

    }catch (NamingException ex) {
      ex.printStackTrace();
    }catch (ClassCastException ex) {
```

```
      ex.printStackTrace();
    }catch (Exception ex){
      ex.printStackTrace();
    }
return empName;
  }
```

Note: The getEmpName(Short) method looks up for the entity bean Employee, creates a remote handle, and calls the methods getFirstName()and getLastName()of Employee entity bean.

Assign entity bean reference to session bean

Since the session bean EmployeeSes is calling the entity bean Employee, an entity bean reference needs to be stored in deployment descriptor. The deployment descriptor tool of JBuilder can be used to create a reference visually.

Double-click **EmployeeSes** under EJBModule TestEntMod from JBuilder project tree and click **EJB Local References** tab at the bottom of the editor pane.

Click **Add** and enter the name as **ejb/Employee** in EJB Local Reference window. Click **OK**.

Click the checkbox **Islink**. The other columns Link, Local Home, and Local should be populated automatically. Just to make sure that the values of the following columns are as follows:

Link: `Employee`
Type: `Entity`
Local home: `com.borland.demo.ejb.EmployeeHome`
Local : `com.borland.demo.ejb.Employee`

See Figure 17 for details.

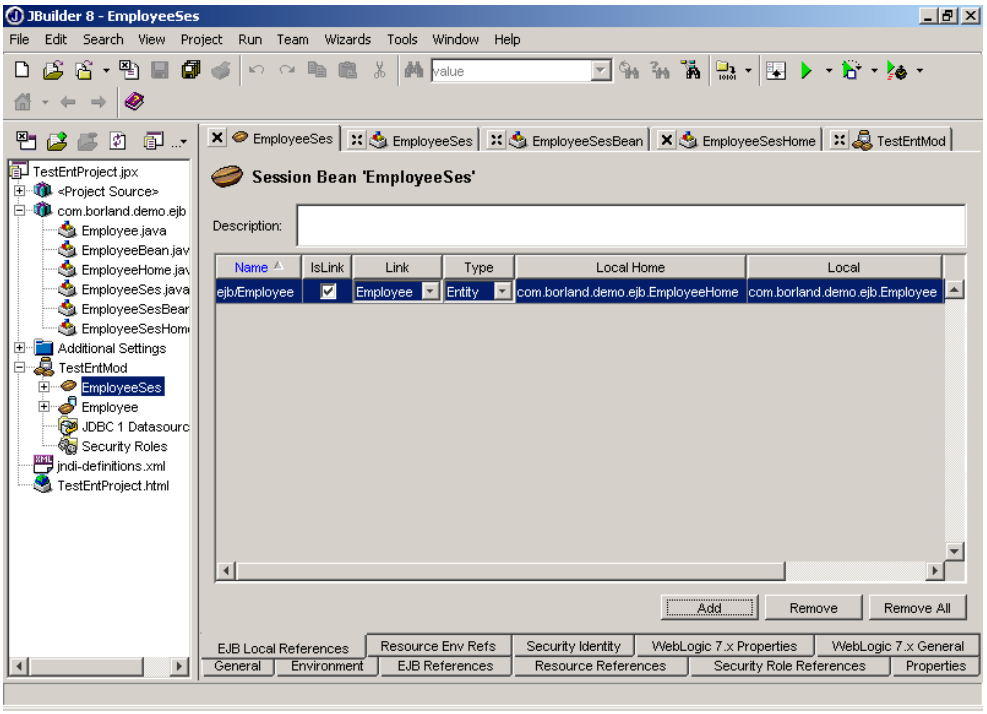

***Figure 17**: Creating an EJB Reference visually using JBuilder deployment descriptor editor tool*

> **Note**: All the utilities in JBuilder are two-way. When you update entries in the deployment descriptor editor, JBuilder automatically changes the DD XML files in the background.

Create data source entries in WebLogic

To create data source entries in WebLogic, the WebLogic Server needs to be started and a connection pool and a transaction data source needs to be created using WebLogic Server Console.

Start WebLogic Server and JDataStore server

Create a runtime configuration to start WebLogic Server within JBuilder. From JBuilder main menubar, click **Run -> Configurations**. The Project Properties window should show up with Run tab selected. Click **New** to create a new runtime configuration. In the Runtime Properties window, enter the following values:

Name:	`Start WebLogic 7.0 [Users can enter any name here]`
Build target:	`<None>`
Type:	`Server`

Click **OK** and now you can see a runtime configuration called **Start WebLogic 7.0** in Project Properties window. Click **OK**.

Click the **Run Project** icon from the JBuilder toolbar and click **Start WebLogic 7.0**. WebLogic Server will start within JBuilder.

To start the JDataStore server, **click Start -> All Programs -> JBuilder 8 Enterprise -> JDataStore Server.**

> **Note**: the JDataStore server has to be started for this example because the data source uses a remote connection to the JDataStore database. You can also connect to the JDataStore database locally, that way you don't have to start the JDataStore server. But the local connection allows only one connection to the JDataStore database. Make sure that the JDataStore server is running all the time during execution of this exercise.

Start WebLogic Server Console. To start the WebLogic Server Console, from JBuilder main menubar, click **Tools -> WebLogic 7.x Admin Console**.

Create a connection pool in WebLogic

If you want to start WebLogic from outside of JBuilder environment for creating a data source, then add jdsserver.jar to the classpath of WebLogic Server. Find **StartWLS.cmd** from <bea_installed_dir>\weblogic700\server\bin and change the **CLASSPATH** as follows:

```
set
CLASSPATH=%JAVA_HOME%\lib\tools.jar;%WL_HOME%\server\lib\weblogic.jar
;%JB_HOME%\lib\jdsserver.jar;%ORACLE_HOME%\jdbc\lib\classes12.jar;%CL
ASSPATH%;
```

From WebLogic Server Console, expand **Services** -> expand **JDBC**.® Click **Connection Pools**. Click **Configure a New JDBC Connection Pool** from the right window. Enter the following parameters to create a connection pool.

Name: `JDSPool`

URL:

`jdbc:borland:dsremote://localhost//E:\Borland\JBuilder8\samples\JDataStore\datastores\employee.jds`

Driver Classname: `com.borland.datastore.jdbc.DataStoreDriver`

Properties (key=value): : `user=Sample`

`password=`

Click **Apply**. See Figure 18 for details.

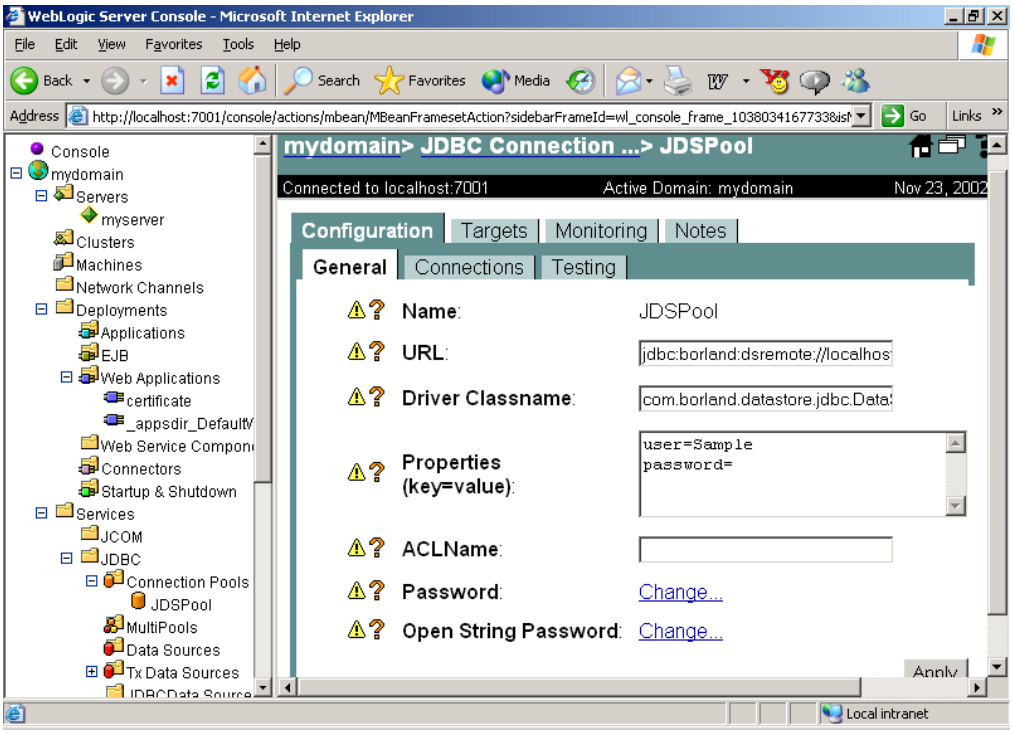

***Figure 18**: WebLogic console showing the connection pool JDSPool*

Click **Connections** from the right window, which is under Configuration tab. Provide the values of the following parameters.

Initial Capacity:
Maximum Capacity:
Capacity Increment:

> **Note**: Provide the values according to your needs. For this exercise, the values for these parameters are 2, 10, and 1 respectively.

See Figure 19 for details.

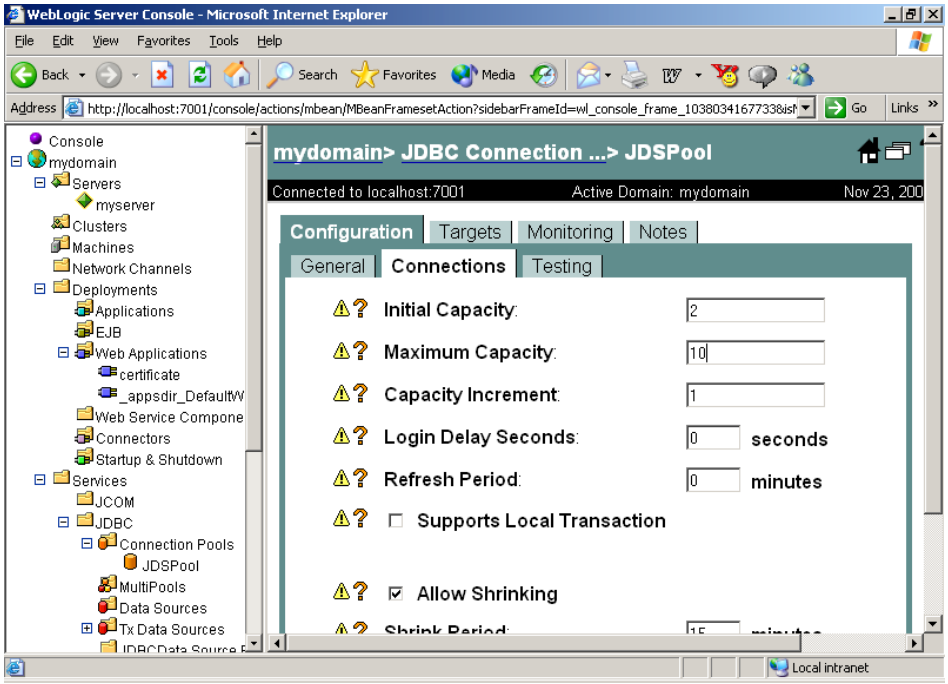

***Figure 19**: WebLogic Console; specifying the connection parameters for the connection pool*

Click **Targets** -> move myserver from **Available** to **Chosen**. Hit **Apply**.

Note: Once you move myserver from Available to Chosen, you can check the JDataStore server for open connections. A successful connection pool creation will create the number of connections in JDataStore, where the number of connections is the Initial Capacity specified in the Connection tab of the connection pool **JDSPool.**

See Figure 20 for a screenshot of JDataStore server.

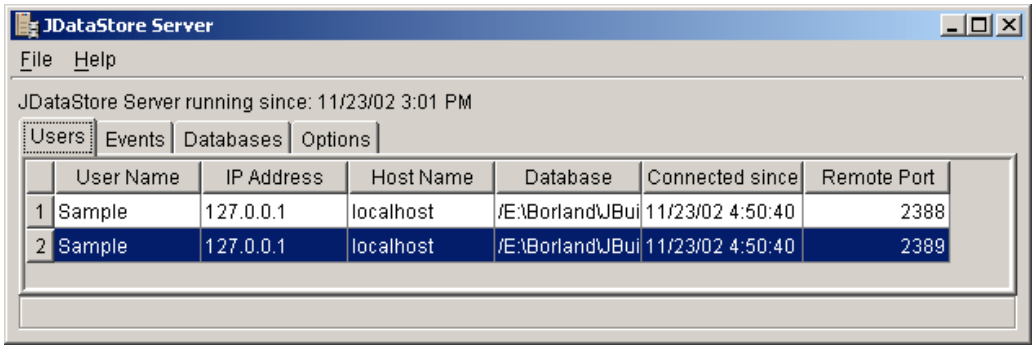
JDataStore Server
File Help
JDataStore Server running since: 11/23/02 3:01 PM
Users Events Databases Options
User Name IP Address Host Name Database Connected since Remote Port
1 Sample 127.0.0.1 localhost /E:\Borland\JBui 11/23/02 4:50:40 2388
2 Sample 127.0.0.1 localhost /E:\Borland\JBui 11/23/02 4:50:40 2389

***Figure 20**: JDataStore server displaying two connections opened by WebLogic 7.0*

Create a transactional data source

From the WebLogic Server Console, expand **Services** -> expand **JDBC**. **Click Tx Data Sources**. Click **Configure new JDBC Tx Data Source** from right window. Enter the following values to create a transactional data source:

Name: `JDS Data Source`

JNDI Name: `JDSDataSource`

Pool Name: `JDSPool`

See Figure 21 for details.

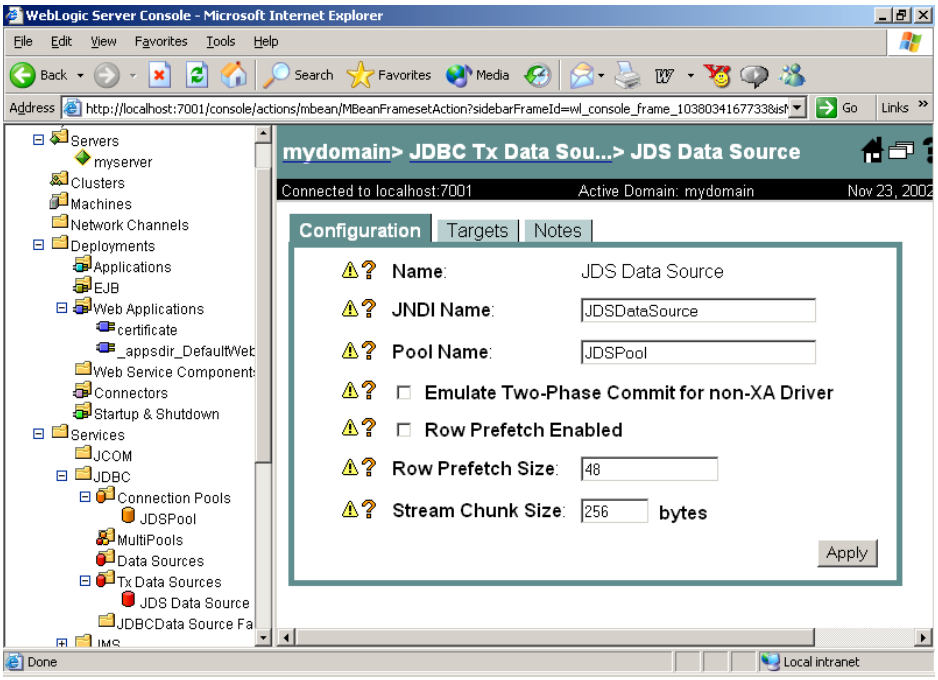

***Figure 21**: WebLogic Console window to creating a data source*

Click **Targets** -> move myserver from **Available** to **Chosen**. Click **Apply**.

Test the data source

To make sure that you have created the data source and pool successfully, stop the WebLogic Server from JBuilder and start it again. If you don't see any errors in the WebLogic Server window of JBuilder, that means the connection pool and data source are created and running successfully. Make sure that the JDataStore server is running.

Deploy and run EJBs in WebLogic

Make the project **(Ctrl + F9)**. This creates the WebLogic specific deployment descriptors in JBuilder. The deployment descriptors for this EJB application are ejb-jar.xml, weblogic-ejb-jar.xml, and weblogic-cmp-rdbms-jar.xml.

If WebLogic is not running within JBuilder, start it by clicking **Run Project** icon**-> Start WeLogic 7.0**. For deploying the EJBModule TestEntMod.jar, right-click **TestEntMod.jar** from project tree. Click **Deploy Options for TestEntMod.jar**. Click **Deploy**.

Now **WebLogic Application Server 7.x Enterprise Deployer** tab opens in JBuilder message pane showing the deployment status. Make sure that there are no deployment errors.

Test the entity bean

To test the entity bean Employee, create a test client to call the session bean EmployeeSes which in turn calls the entity bean Employee. The entity bean Employee gets the data from the database, sends to the session bean and the session bean sends the data to the test client. Refer to the previous section Test the session bean in the chapter Working with session beans to get help on how to create a test client.

For this exercise, the name of the test client is **EmployeeSesTestClient** and the package name for this test client is **com.borland.demo.ejb**. JBuilder automatically creates a runtime configuration called **EmployeeSesTestClient** for running the test client within JBuilder.

Modify the main() method of the test client EmployeeSesTestClient.java as follows:

```
public static void main(String[] args) {
    EmployeeSesTestClient client = new EmployeeSesTestClient();
    try {
      client.create();
      String empName = client.getEmpName(new Short("4"));
      System.out.println("Employee Name from the Test Client is = " +
          empName);
    }catch (Exception ex) { ex.printStackTrace(); }
}
```

Run the test client

From the JBuilder toolbar, click the **Run Project** icon. Click **EmployeeSesTestClient**. Now the test client EmployeeSesTestClient.java will run in JBuilder in a separate tab called EmployeeSesTestClient. A successful execution will return the name `Bruce Young` from the Employee entity bean. See Figure 22 for details.

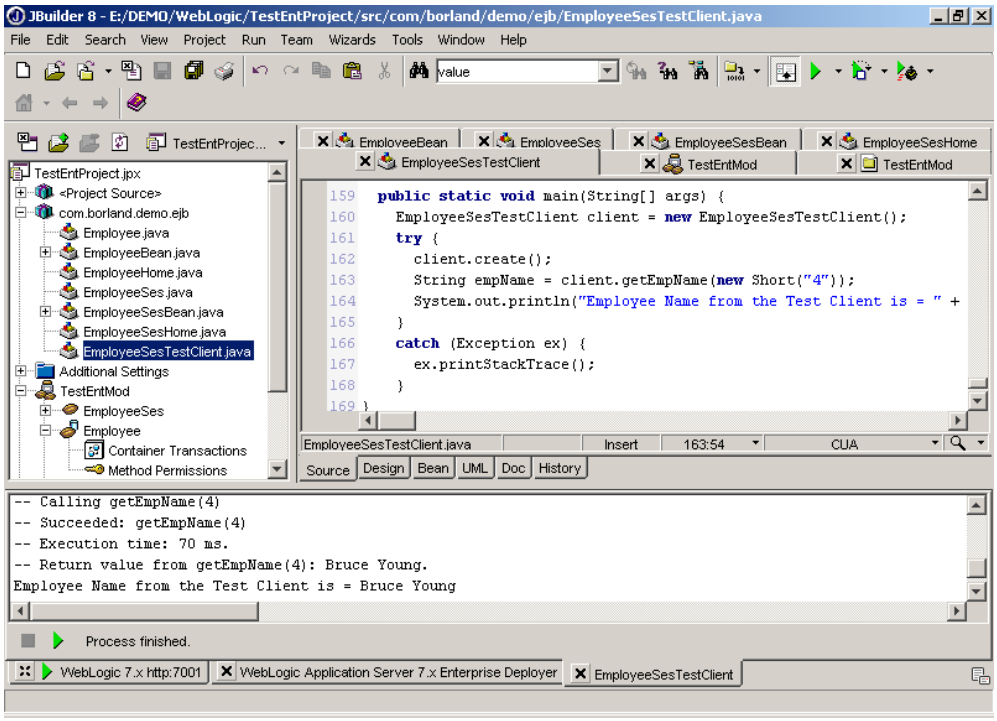

***Figure 22**: The test client displaying the outputs from running EJBs in WebLogic 7.0*

Remote debugging of EJBs

JBuilder can remotely debug an already running application in a remote server, in this case WebLogic Server. To remotely debug an application, the application server has to be started in debug mode.

Note: For this exercise, let's take the exercise created in the previous chapter "Working with entity beans." Also, this exercise assumes that you have already created the entity bean, session bean, test client, and deployed to WebLogic Server.

Provide remote debug options

From Internet Explorer, go to **E:\bea\user_projects\mydomain**. Copy **startWebLogic.cmd** to **startRemoteWeblogic.cmd** in the same directory and make the following changes to startRemoteWeblogic.cmd:

```
set JAVA_OPTIONS=-Xdebug -Xnoagent -
Xrunjdwp:transport=dt_socket,server=y,address=5555,suspend=n -
Dweblogic.security.SSL.trustedCAKeyStore=E:\bea\weblogic700\server\li
b\cacerts
```

Note: You can also optionally provide your user name and password at the following two statements. That way you don't have to provide the user ID and password at the time of running WebLogic Server.

```
set WLS_USER=
set WLS_PW=
```

Add jdsserver.jar to the classpath of the remote WebLogic Server. Find StartWLS.cmd from <remote_bea_installed_dir>\weblogic700\server\bin and change the CLASSPATH as follows:

```
set
CLASSPATH=%JAVA_HOME%\lib\tools.jar;%WL_HOME%\server\lib\weblogic.jar
;%JB_HOME%\lib\jdsserver.jar;%ORACLE_HOME%\jdbc\lib\classes12.jar;%CL
ASSPATH%;
```

Create runtime configuration for remote debugging

From the JBuilder menubar, click **Run -> Configurations**. The Project Properties window with Run tab selected should pop up. Click **New**. Enter the following configuration values in the Runtime Configuration Properties window.

Name: `RemoteDebug`

Build target: `<None>`

Type: `Server`

Click **Debug** tab from Runtime Configuration Properties window.

Check the **Enable Remote Debugging** checkbox, and click **Attach** in the Remote Settings panel. Now the Type and Address textbox of Transport panel should be active. Enter the address as **5555**.

See Figure 23 for details.

> **Note**: The address specified here must be the same as the port number specified in the WebLogic server as debug options. JBuilder will open a socket connection to the WebLogic Server and attach to WebLogic.

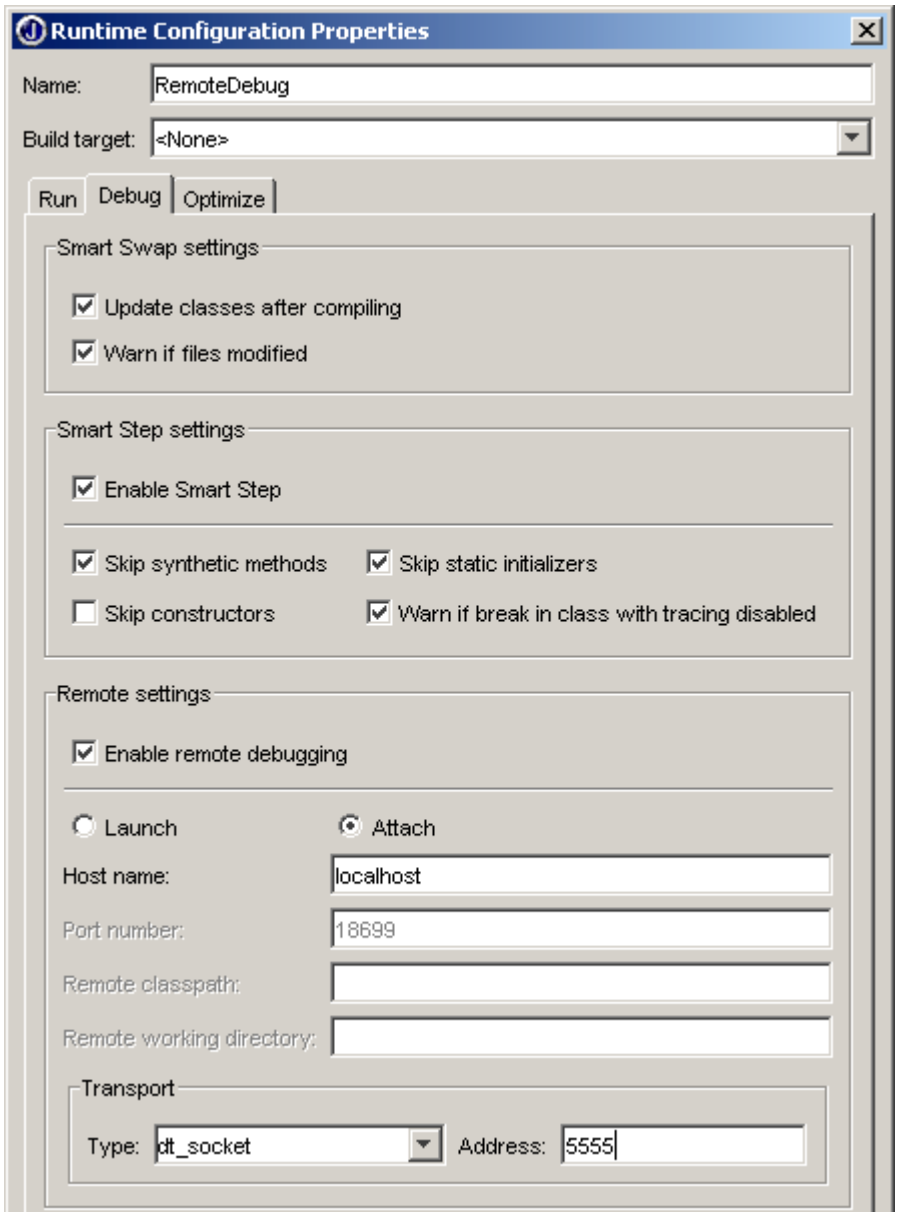

***Figure 23**: Runtime Configuration Properties window with remote debug parameters to remote debug EJBs*

Assign breakpoints

Assign breakpoints to the entity bean and session bean to test the remote debugging as follows:

Breakpoint 1
File name: EmployeeSessionBean.java
Line Description : javax.naming.Context context = new javax.naming.InitialContext();

Breakpoint 2
File name: EmployeeBean.java
Line Description : public abstract java.lang.String getFirstName();

Run WebLogic Server in debug mode outside of JBuilder

Start a MS-DOS window, go to directory **E:\bea\user_projects\mydomain** and type **startRemoteWebLogic.**

If WebLogic Server has started successfully, you will see the following message from WebLogic Server window:

```
<Notice> <WebLogicServer> <000360> <Server started in RUNNING mode>
```

See Figure 24 for details.

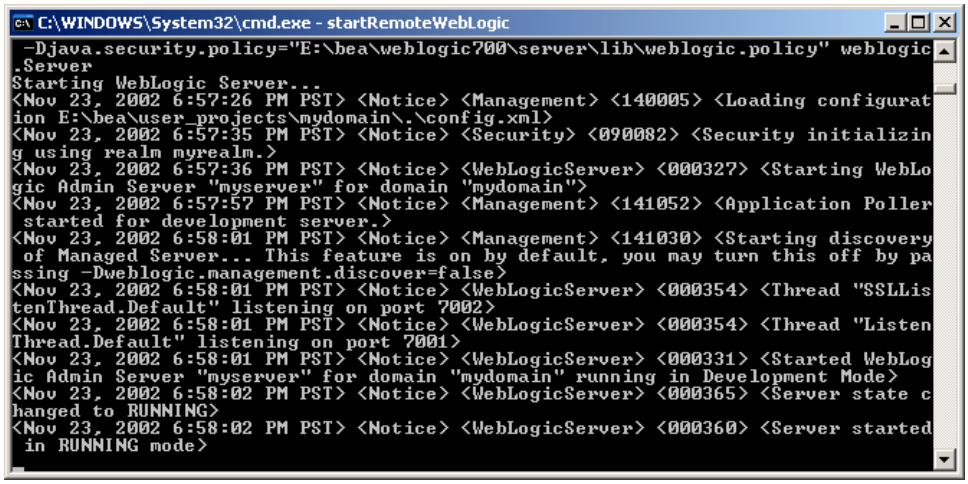

***Figure 24**: WebLogic Server 7.x running in remote mode outside of JBuilder*

Note: For this exercise, WebLogic Server is started in the same machine as JBuilder 8 just to explain the steps. You can start WebLogic Server in debug mode in any machine in the network, and JBuilder will be able to connect to the remote machine using the socket address.

Attach JBuilder to WebLogic remotely

To attach JBuilder to WebLogic Server remotely, from the JBuilder toolbar, click **Debug Project** icon. Click **RemoteDebug**.

Now a tab with name **localhost:5555** will be opened in the JBuilder message pane.

See Figure 25 for details.

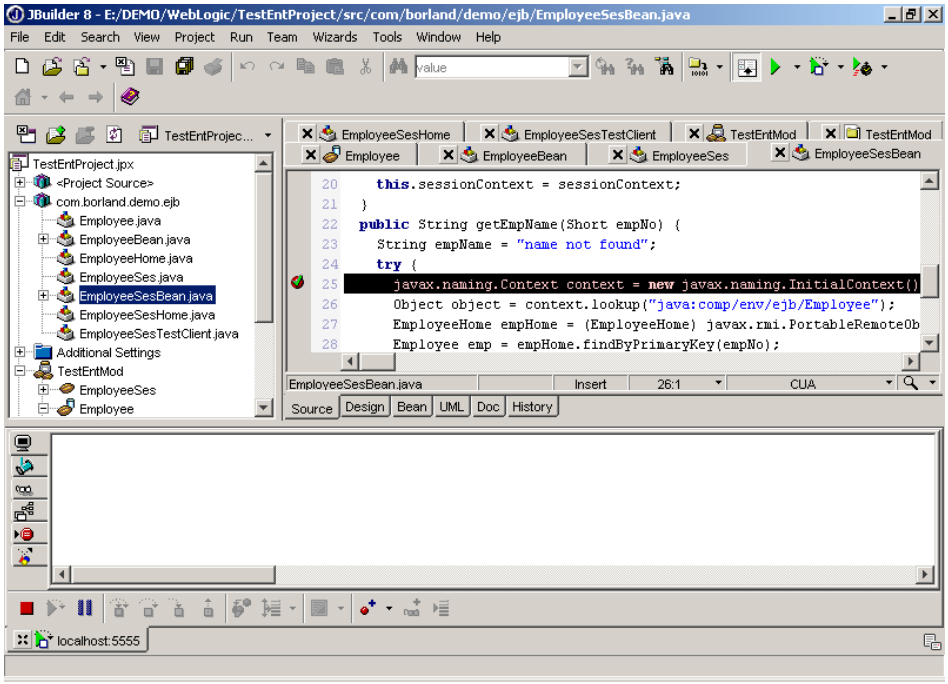

***Figure 25**: JBuilder opened a connection to remote WebLogic Server*

> **Note**: JBuilder puts a green check mark against all valid breakpoints and a red cross mark against all non-valid breakpoints. Click EmployeeBean.java which shows a red cross mark against the breakpoint at line 29 because the breakpoint is not valid.

Run the test client to debug the EJBs

From the JBuilder toolbar, click **Run** icon ->**EmployeeSesTest Client**. Now the test client starts in another tab in JBuilder and the debugger stops at the breakpoint in EmployeeSesBean.java.

See Figure 26 for details.

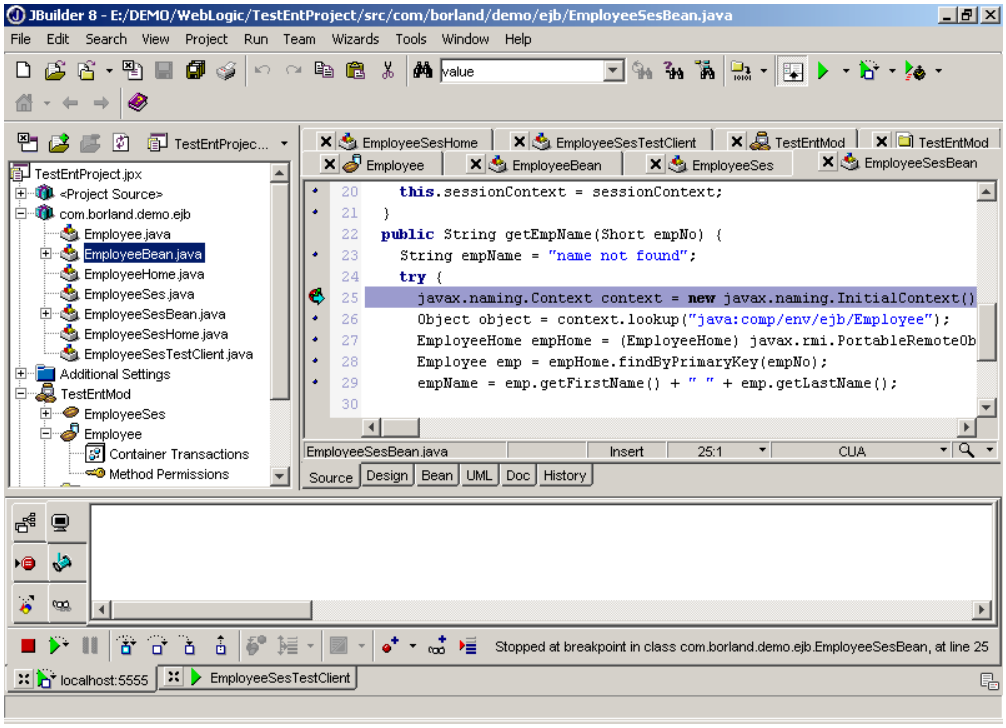

***Figure 26**: JBuilder remotely debugging running EJBs. Debugger stopped at the breakpoint*

Now you can follow your normal debugging process like step over, step into, step out, etc.

> **Note**: Remote debugging process is slower that the normal run process. This is because the JVM has to operate in a debug mode which is slower than the normal run mode.

Working with Message Driven Beans

JBuilder provides easy-to-use wizards to create Message Driven Beans (MDB) applications easily. To work with MDB, the JMS configurations needs to be created in WebLogic Server.

The required configurations are as follows:

- Create a Connection Factory
- Create a JMS store
- Create a JMS Server
- Create a Destination (Queue, Topic)

To learn more about creating JMS application using WebLogic Server, visit the link http://e-docs.bea.com/wls/docs70/jms/implement.html

Create JMS Configurations in WebLogic®

Create a Connection Factory

Start **WebLogic Server** and open **WebLogic Server Console**. Expand **mydomain -> Services -> JMS** -> click **Connection Factories**. From right-hand pane, click **Configure a new JMS Connection Factory**.

Enter the following values in the right-hand side window:
Name: `DemoJMSConnectionFactory`
JNDIName: `DemoJMSConnectionFactory`

See Figure 27 for details.

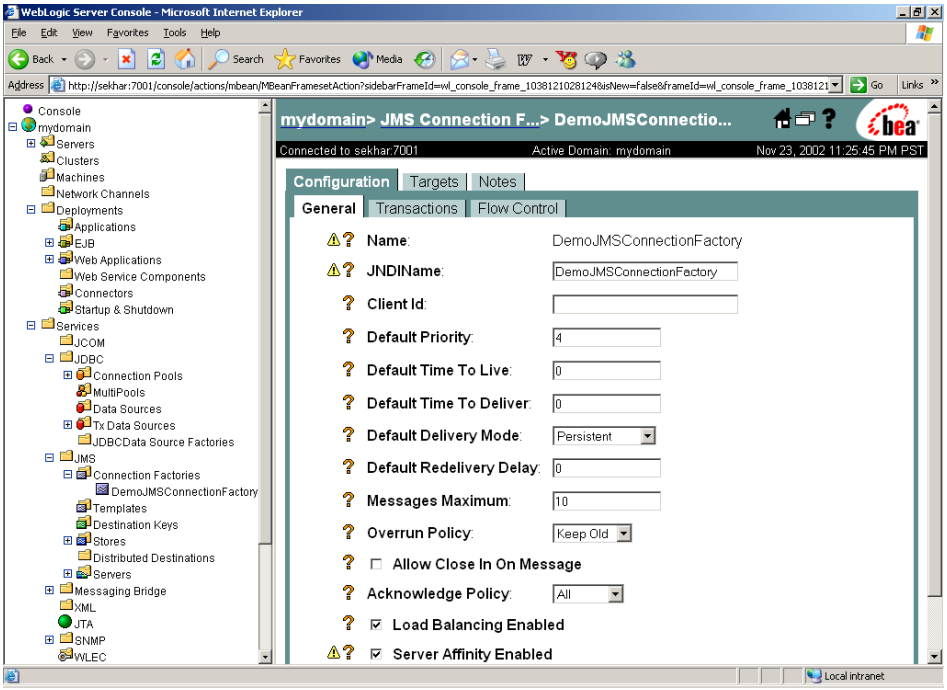

***Figure 27**: Creating a JMS Connection Factory using WebLogic Server Console*

Leave other default values and click **Create**. Click **Target** tab and move myserver from **Available** to **Chosen**.

Create a JMS Store

Two types of JMS Configurations can be created in WebLogic: **JMSJDBC Store** and **JMSFile Store**. For this exercise, let's create a JMSFileStore.

From WebLogic Server Console, expand **JMS** under Services and click **Stores**. Click **Configure a new JMSFile Store**.

Enter the following values in the right-hand side window:

Name	`DemoJMSFileStore`
Directory	`E:\bea\DEMOJMSFileStore`

Make sure that you create the specified directory, [i.e. E:\bea\DEMO JMS FileStore] before entering here. Otherwise the JMS Store cannot be created.

See Figure 28 for details.

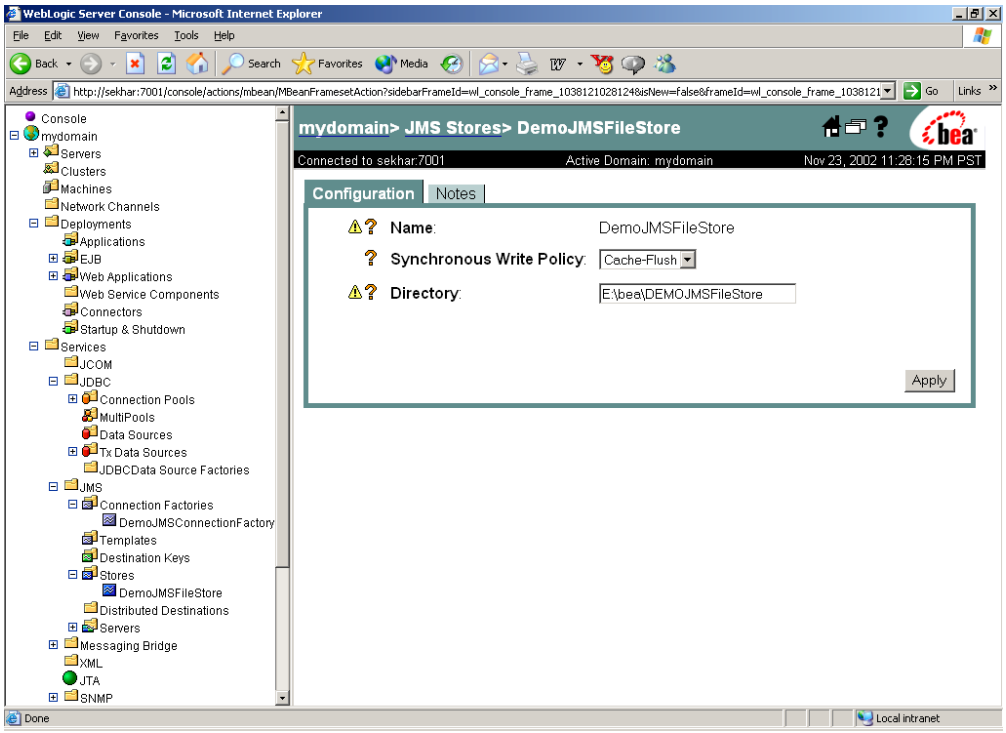

***Figure 28**: Creating a JMS File Store using WebLogic Server Console*

Create a JMS Server

From WebLogic Server Console, expand **JMS** under Services and click **Servers**. Click **Configure a new JMSServer**.

Enter the following values in the right-hand side window:

Name: `DemoJMSServer`

Store: `DemoJMSFileStore`

Leave other defaults. Click **Targets** tab and select **myserver** as Target.

See Figure 29 for details.

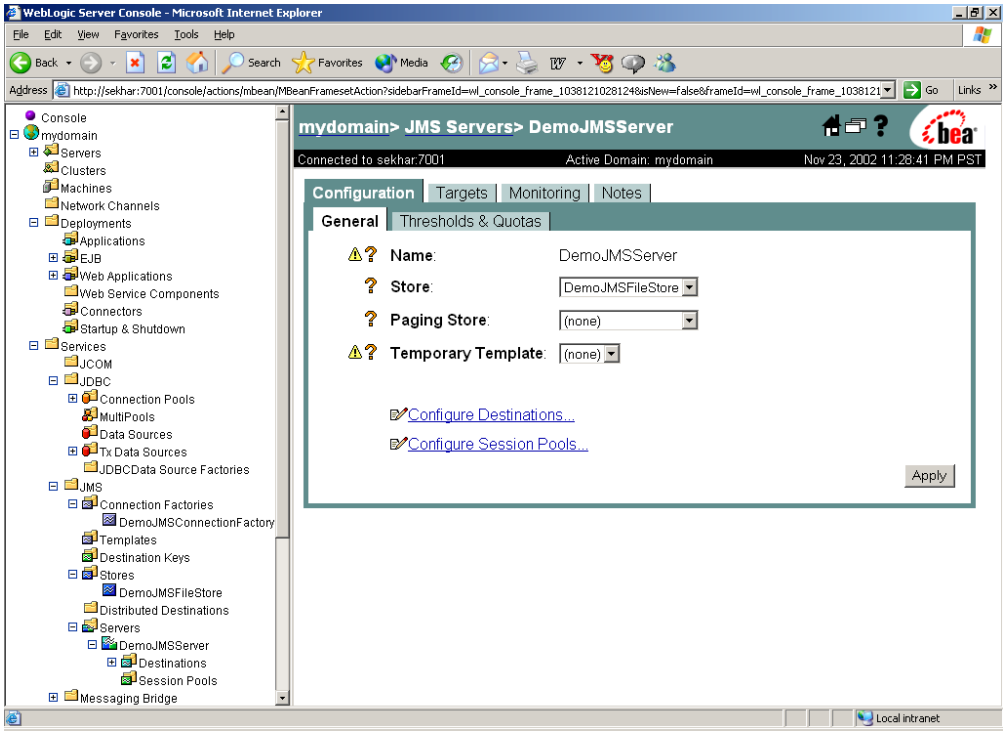

***Figure 29**: Creating a JMS Server using WebLogic Server Console*

Create a Queue

From WebLogic Server Console, expand **JMS** under Services. Expand **Servers** -> **DemoJMSserver -> Destinations** ->Click **Configure a new JMSQueue**.

Enter the following values in the right-hand side window:

Name: `DemoJMSQueue`

JNDIName: `DemoJMSQueue`

Leave other defaults. Click **Create**. See Figure 30 for details.

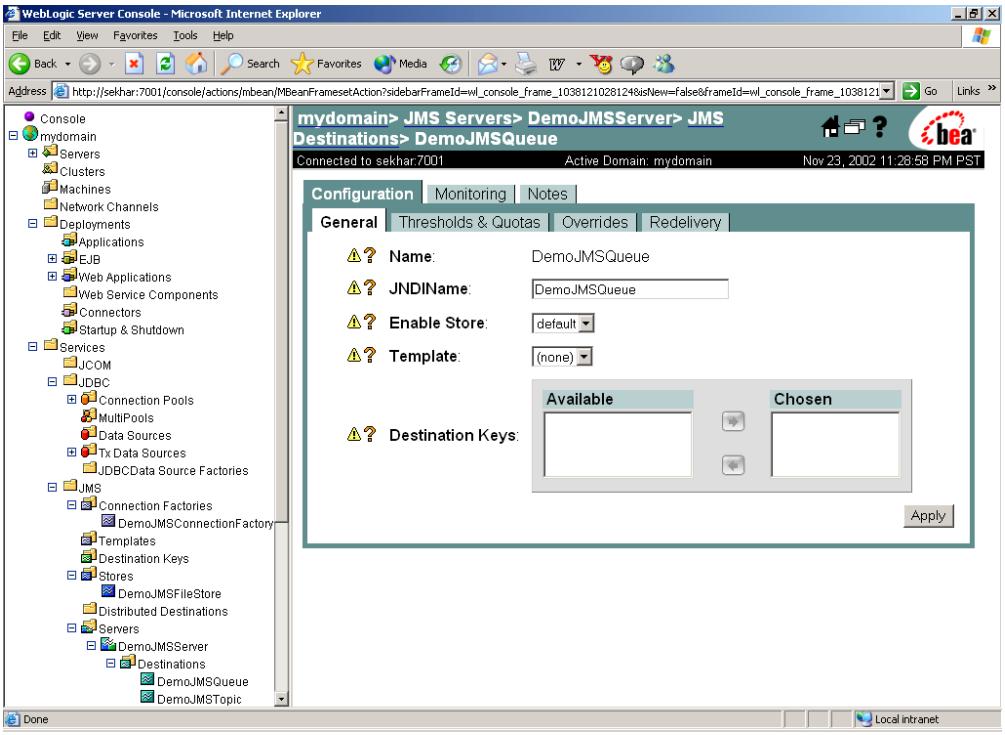

***Figure 30**: Creating a JMS Server using WebLogic Server Console*

Create a Message Driven Bean

Create a project, assign WebLogic Server 7.x as the target application server for the project, and create an EJB module. To get help on how to perform the above operations, refer to the previous chapter "Working with session beans."

For this exercise, enter the name of the project as **TestJMSProject** in the directory **E:\Demo\WebLogic\TestJMSProject** and enter the EJB module name as **TestJMSMod**.

To create a MDB, double-click **EJBModule TestJMSMod**, right-click on the **editor pane**, click **Create EJB -> Message-Driven Bean**.

Enter the following values in the Bean Properties window:

Name:	`TestJMS`
Transaction Type:	`Container`
Destination Name:	`DemoJMSQueue`
Destination Type:	`javax.jms.Queue`
Initial pool size:	`2`
Maximum pool size:	`10`

Connection factory name : `demoJMSConnectionFactory`

See Figure 31 for details.

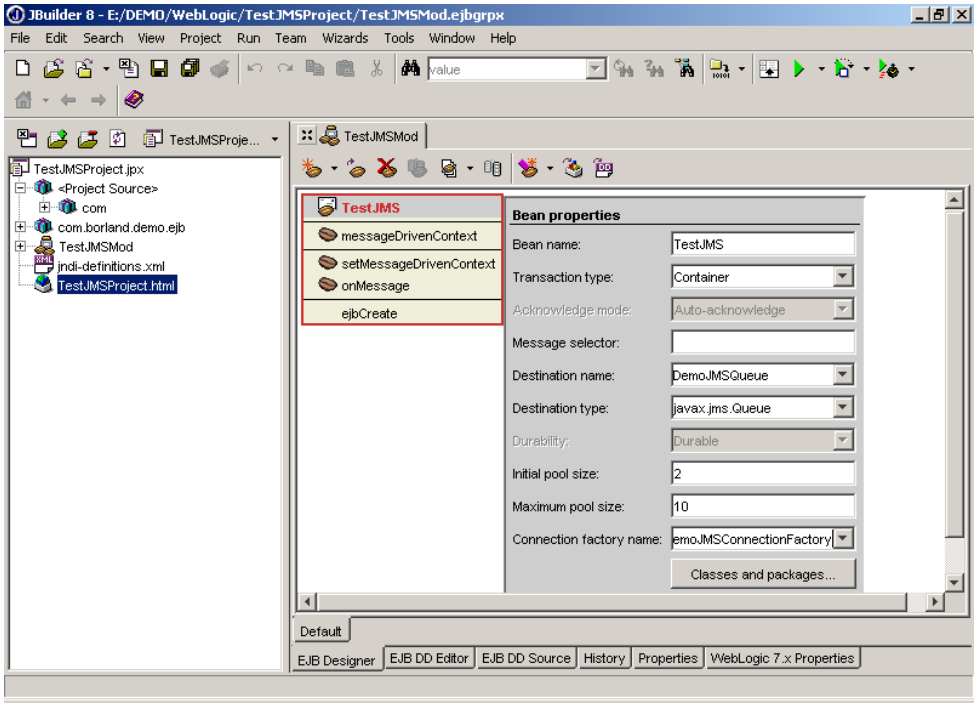

***Figure 31**: JBuilder created a MDB visually and provided a GUI to configure the MDB*

Right-click **the MDB TestJMS** and click **View bean source**. Change the onMessage() method of TestJMSBean.java as follows:

```
public void onMessage(Message msg) {
    try {
      TextMessage txtMessage = (TextMessage) msg;
      System.out.println("Message from EJB TestJMS  = " +
txtMessage.getText());
    }
    catch (JMSException ex) {
    }
  }
```

> **Note**: The on Message (message) method gets executed when the MDB receives a message. In this case, the on Message (message) method will print the received message.

Make the project and make sure that there are no build errors.

Deploy and run the MDB

Create a runtime configuration to start the WebLogic Server within JBuilder. From JBuilder main menubar, click **Run -> Configuration** -> click **New**. In the Runtime Configuration Properties window enter name as **Start WebLogic 7.0** and Build Target as <**None**>.

Start WebLogic Server within JBuilder (click **Run** icon from JBuilder toolbar-> click **Start WebLogic 7.0**).

Deploy TestJMSMod to WebLogic Server (right-click **TestJMSMod** -> click **Deploy options for TestJMSMod.jar** -> click **Deploy**).

Test the MDB

To test the MDB TestJMS, create a test client. A test client can be created by clicking **File -> New** -> **General** tab of **Object Gallery -> Class** -> Click **OK**.

Enter the name of the class as Client. Change the contents of Client.java in JBuilder editor pane as follows:

Client.java

```
package com.borland.demo.ejb;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.jms.Queue;
import javax.jms.QueueConnection;
import javax.jms.QueueConnectionFactory;
import javax.jms.QueueSender;
import javax.jms.QueueSession;
import javax.jms.Session;
import javax.jms.TextMessage;
import javax.naming.NamingException;
import java.util.Hashtable;
public class Client {
  public final static String
      JNDI_FACTORY="weblogic.jndi.WLInitialContextFactory";
  public final static String URL = "t3://localhost:7001";
  private static final String QUEUE_CONNECTION_FACTORY =
      "DemoJMSConnectionFactory";
  private static final String QUEUE = "DemoJMSQueue";
  private static InitialContext getInitialContext(String url)
      throws NamingException
    {
      Hashtable env = new Hashtable();
      env.put(Context.INITIAL_CONTEXT_FACTORY, JNDI_FACTORY);
      env.put(Context.PROVIDER_URL, url);
      return new InitialContext(env);
    }
  public static void main(String[] args) throws Exception {
    Context ctx = getInitialContext(URL);
    QueueConnectionFactory qConnFact =
       (QueueConnectionFactory) ctx.lookup(QUEUE_CONNECTION_FACTORY);
```

```
    QueueConnection qConn = qConnFact.createQueueConnection();
    QueueSession qSess = qConn.createQueueSession(false,
        Session.AUTO_ACKNOWLEDGE);
    Queue q = (Queue) ctx.lookup(QUEUE);
    QueueSender qSend = qSess.createSender(q);
    TextMessage txtMsg = qSess.createTextMessage("MDB
        Testing Successful");
    System.out.println("Sending a message to queue");
    qSend.send(txtMsg);
    qConn.close();
    System.exit(0);
  }
}
```

Build the Client.java file and make sure that there are no compilation errors. Now run Client.java by right-clicking **Client.java** -> **Run using defaults**.

On Successful execution the following message should appear in Client tab:

```
Sending a message to queue
```

Also the following message should appear in the server tab (WebLogic 7.x http:7001):

```
Message from EJB TestJMS  = MDB Testing Successful
```

See Figure 32 for details.

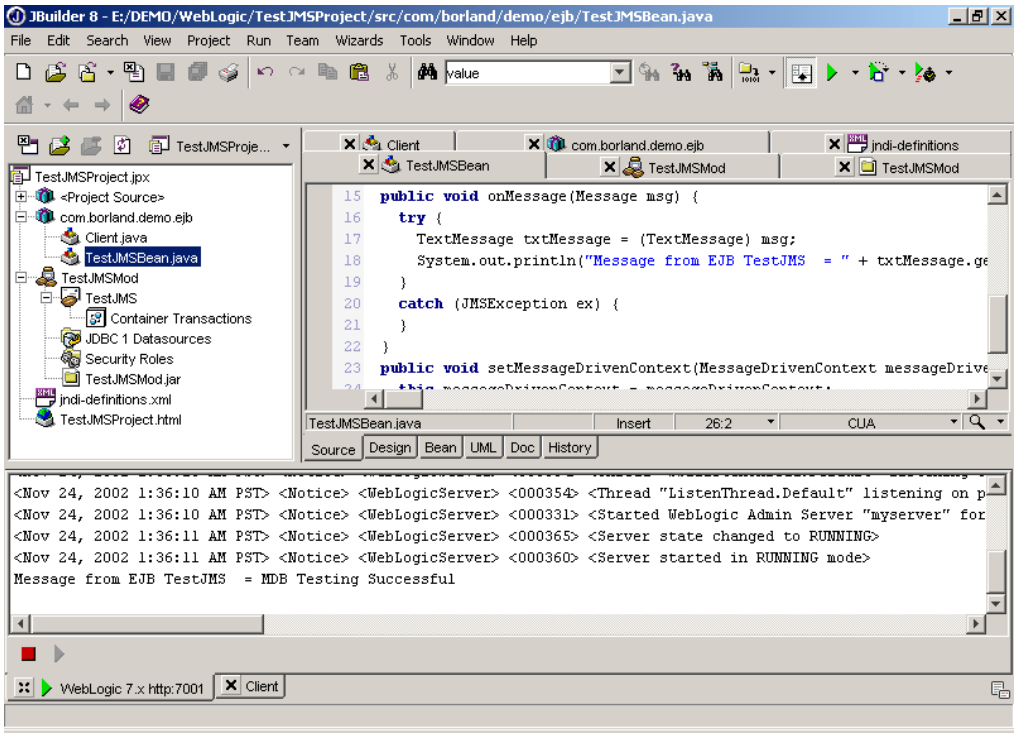

***Figure 32**: JBuilder showing successful execution of MDB in WebLogic Server running within JBuilder*

Troubleshooting

This troubleshooting section highlights the errors that may cause problems during the deployment of session beans and running of the test client. If you have problems starting WebLogic 7.0 within JBuilder 8, refer to the paper "Web Development Using JBuilder 8 and WebLogic 7.0" from the Borland Web site.

Error 1: NameNotFoundException from test client

Error description

```
javax.naming.NameNotFoundException: Unable to resolve 'TestSes'
Resolved: ''Unresolved:'TestSes'; remaining name 'TestSes'
```

When did this happen

I have deployed my EJB JAR file. Now, I am running my test client, and it fails with the above error.

Cause

The test client is unable to find the JNDI name of the EJB.

Solution

Check the deployment of EJB JAR file again. The deployment of TestSesMod may have failed. To check the deployment, start WebLogic Server Console and find the EJB module name TestSesMod from mydomain -> Deployments ->EJB.

Click the EJB Module **TestSesMod** from the from the right-hand pane of WebLogic Server Console , click **Deploy**. If the JAR file is deployed properly, you will see "true" under Deployed column. Also click **Monitoring** tab, and **click Monitor all Stateless EJBRuntimes**.

In case of unsuccessful deployment, try to re-deploy again.

Error 2: Test client run failed

Error description

```
javax.naming.CommunicationException. Root exception is
java.net.ConnectException: t3://localhost:7001: Destination
unreachable; nested exception is:
java.net.ConnectException: Connection refused: connect; No available
router to destination
```

When did this happen

I have deployed my EJB JAR file successfully. When I run my test client, I got the above error message from the Test Client tab of JBuilder message pane.

Cause

The test client is unable to communicate with WebLogic Server.

Solution

Note that the PROVIDER_URL to connect to WebLogic Server is hardcoded in the test client. The hardcoded value is t3://localhost:7001. If, for any reason, you have started your WebLogic Server in another hostname or port number, change this value in the test client TestSesTestClient.java to match the new values.

Error 3: Cannot create connection pool

Error description

```
java.lang.NullPointerException
at


weblogic.management.console.utils.MBeans.getMBeanClassNameFor(MBeans.
java:1153)
```

When did this happen

I tried to create my JDSPool using WebLogic Console. I entered the configuration parameters and then tried to assign this JDSPool to myserver. (To assign to myserver, I clicked Targets and moved myserver from Available to Chosen.) I see the above error message in the WebLogic Server Console.

Cause

The URL name, Driver name, user ID, or password entered in the JDSPool Configuration may not be correct. If those entries are right, then the Driver Classes may not have loaded during the WebLogic Server startup.

Solution

Go back to JBuilder message pane and check the error messages from WebLogic Server.

If the error message is:

```
<Error> <JDBC> <001060> <Cannot startup connection pool "JDSPool"
java.sql.SQLException: No suitable driver>
```

From the WebLogic Console, click **Targets**, move myserver from Chosen to **Available**. Click **Apply**. Click **Configurations** and correct the URL name. Click **Apply**. Again,click **Targets**, move myserver from Available to **Chosen** and see if the error message is gone.

If the error message is:

```
<Error> <JDBC> <001060> <Cannot startup connection pool "JDSPool"
weblogic.common.ResourceException: Cannot load driver class:
om.borland.datastore.jdbc.DataStoreDriver>”
```

From WebLogic Console, click **Targets**, move myserver from Chosen to **Available**. Click **Apply**. Click **Configurations** and correct the Driver name, click **Apply**. Again, click **Targets**, move myserver from Available to **Chosen** and see if the error message is gone.

If the error message is:

```
<Error> <JDBC> <001060> <Cannot startup connection pool "JDSPool"
weblogic.common.ResourceException: weblogic.common.ResourceException:
Could not create pool connection. The DBMS driver exception was:
java.sql.SQLException: You do not have STARTUP rights for this
JDataStore
```

From the WebLogic Console, click **Targets**, move myserver from Chosen to **Available**. Click **Apply**. Click **Configurations** and correct the username and password, click **Apply**. Make sure that you provide the user ID and password information as follows:

```
user=
password=
```

Click **Targets**, move myserver from Available to **Chosen** and see if the error message is gone.

If the error message is:

```
<Error> <JDBC> <001060> <Cannot startup connection pool "JDSPool"
weblogic.common.ResourceException: Cannot load driver class:
com.borland.datastore.jdbc.DataStoreDriver>
```

From the WebLogic Console, click **Targets**, move myserver from **Chosen** to **Available**, click **Apply**. Check the WebLogic Server configuration from JBuilder. From the JBuilder menubar, click **Tools -> Configure Servers**. Click **WebLogic Server 7.x** from the left panel and check if you have added the JDataStore driver package jdsserver.jar user in the Class tab in the right panel.

If you haven't added them, click **Add** and add jdsserver.jar from **<JBuilder8_Installed_Dir>\lib** directory.

Restart WebLogic Server, click **Targets for JDSPool** from the WebLogic Server Console, move myserver from **Available** to **Chosen**, and see if the error message is gone.

Error 4: Cannot create data source

Error description

```
java.lang.NullPointerException at
weblogic.management.console.utils.MBeans.getMBeanClassNameFor(MBeans.
java:1153)
```

When did this happen

I have created my connection pool called **JDSPool** successfully using the WebLogic Server Console. Now trying to create my data source called **JDSDataSource**, I have entered the configuration parameters and then tried to assign this JDSatasource to myserver. (To assign to myserver, I clicked Targets and moved myserver from Available to Chosen.) I see the above error message in the WebLogic Server Console.

Cause

The pool name entered may not be correct or JDSPool may not be active.

Solution

Go back to the JBuilder message pane and see the error message from the WebLogic Server window.

If the error message is:

```
<Error> <JDBC> <001059> <Error during Data Source creation:
weblogic.common.ResourceException: DataSource(JDSDataSource) can't be
created with non-existent Pool (connection or multi) (JDSPoo)
```

From the WebLogic Server Console, click **JDSPool**, click **Targets**, move myserver from Chosen to **Available**, click **Apply**. Click **configurations** and correct **pool name**, click **Apply**. Again, click **Targets**, move myserver from Available to **Chosen** and see if the error message is gone.

If the error message still exist, then check the Pool Name for JDSDataSource and make sure that it it correct.

Error 5: Can't start connection pool in WebLogic

Error description

```
<Error> <JDBC> <001060> <Cannot startup connection pool "JDSPool"
weblogic.common.ResourceException: weblogic.common.ResourceException:
Could not create pool connection. The DBMS driver exception was:
java.net.ConnectException: Connection refused: connect.
```

When did this happen

I have created my connection pool **JDSPool** and data source **JDSDataSource** successfully. When I started my WebLogic Server, I got the above error.

Cause

WebLogic Server is trying to connect to the database, and it is failing. Possible reason is the database may not be available.

Solution

Start the **JDataStore Server**. **Click Start -> All Programs -> JBuilder 8 Enterprise -> JDataStore Server** to start the Server. After starting JDataStore Server, **stop WebLogic Server** and start it again to see if the error is gone.

Error 6: Test client not returning employee name

Error description

```
Employee Name from the Test Client is = name not found
```

When did this happen

I have successfully created connection pool and a data source using the WebLogic Server Console. I have started my WebLogic Server and JDataStore Server and have successfully deployed my EJB JAR file to WebLogic. I ran my test client, and the test client got the following message from the EmployeeSesTestClient tab.

Cause

Your test client is communicating properly with the session bean EmployeeSes. But the session bean EmployeeSes is not connecting to the entity bean Employee.

Solution

Go back to the JBuilder message pane and see the error message from WebLogic Server tab.

If the error message is:

```
javax.naming.NameNotFoundException: Unable to resolve
'app/ejb/TestEntMod.jar#EmployeeSes/comp/env/ejb/Employee' Resolved:
'app/ejb/TestEntMod.jar#EmployeeSes/comp/env' Unresolved:'ejb' ;
remaining name 'ejb/Employee
```

Check if your entity bean reference from session bean is correct. Refer to section “Assign entity bean reference to session bean.”

Error 7: EJB JAR deployment failed

Error description

```
Exception:weblogic.management.ApplicationException: activate failed
for TestEntMod
      Module: TestEntMod  Error: Exception activating module:
EJBModule(TestEntMod,status=PREPARED)
Unable to deploy EJB: Employee from TestEntMod.jar:

weblogic.ejb20.WLDeploymentException: The DataSource with the JNDI
name: JDSDataSource could not be located. Please ensure that the
DataSource has been deployed successfully and that the JNDI name in
your EJB Deployment descriptor is correct.
```

When did this happen

I have successfully created a connection pool and a data source using the WebLogic Console. I have started my WebLogic Server and JDataStore server and when trying to deploy my EJB JAR file, I got the above error from WebLogic Server 7.x Enterprise Deployer tab from JBuilder.

Cause

The data source name created in WebLogic and the data source name specified in the Database Schema Provider window of JBuilder (while Importing Schema from Database) does not match.

Solution

Check the data source name from the WebLogic Server Console. From the JBuilder Project tree, double-click entity bean **Employee** and click **WebLogic 7.x** general tab from the editor pane. Change the data source name to match with the data source name created in WebLogic.

Save all, **make** the project, **and re-deploy** the EJB JAR file again.

Error 8: WebLogic remote server failed to start

Error description

```
 <Server failed during initialization.
Exception:weblogic.management.configuration.ConfigurationException:
undefined mbean reference:
mydomain:Name=JBuilder_DefaultWebApp,Server=myserver,Type=WebAppCompo
nentweblogic.management.configuration.ConfigurationException:
undefined mbean reference:
mydomain:Name=JBuilder_DefaultWebApp,Server=myserver,Type=WebAppCompo
nent
```

When did this happen

The error occurred when I tried to run startRemoteWebLogic.cmd from the command prompt.

Cause

The error occurred because of the unreferenced Web application JBuilder_DefaultWebApp in the WebLogic Server. JBuilder creates this application in WebLogic and sometimes doesn't delete it properly.

Solution

From the WebLogic Console, expand **Deployments** -> expand **Web Applications**. Delete **JBuilder_DefaultWebApp** from the right window.

Also check the config.xml from E:\bea\weblogic700\server\bin and delete the following lines.

```
<WebServer DefaultWebApp="JBuilder_DefaultWebApp"
            LogFileName=".\myserver\access.log" LoggingEnabled="true"
Name="myserver"/>
```

Error 9: JBuilder attachment to WebLogic failed

Error description

```
-IOException attaching Remote Java debug Process –
java.net.ConnectException : Conection refused: connect
```

When did this happen

The error occurred when I tried to attach JBuilder with a running WebLogic Server.

Cause

The error occurred because either WebLogic is not running properly, or the debug parameters provided to start WebLogic are not correct.

Solution

Check if WebLogic is running. If it is running, then check the debug parameters. If the debug parameters are also correct, then check JBuilder Remote Debug configuration. From the JBuilder menubar, click **Run -> Configuration**. Edit the **Remote Debug** entry. Click **Debug** tab and check the address. Make sure that the port number provided in this address is the same as the address provided in WebLogic debug options.

Useful links

JBuilder product documentation

http://info.borland.com/techpubs/jbuilder/jbuilder8/index1280x1024-ent.html

Optimizeit Suite product documentation

http://info.borland.com/techpubs/optimizeit/optimizeit5/index1280x1024.html

JBuilder tools and components

http://info.borland.com/jbuilder/resources/jbtools.html

JBuilder OpenTools

http://codecentral.borland.com/codecentral/ccweb.exe/prodcat?prodid=3&catid=11

JBuilder developers support

http://info.borland.com/devsupport/jbuilder/

JBuilder Frequently Asked Questions

http://community.borland.com/all/0,1435,c|3|10,00.html

JBuilder white papers

http://community.borland.com/all/0,1435,c|3|13,00.html

JBuilder certification

http://www.borland.com/services/certification/index.html

Training on Borland products

http://www.borland.com/services/training/course_descriptions.html

Borland newsgroups

http://info.borland.com/newsgroups/

Feedback and suggestions

Please send your feedback and suggestions to :

Sudhansu Pati, Systems Engineer

spati@borland.com

100 Enterprise Way, Scotts Valley, CA 95066-3249

www.borland.com | 831-431-1000 | Fax: 831-431-4142

Copyright © 200 Borland Software Corporation. All rights reserved. All Borland brand and product names are trademarks or registered trademarks of Borland Software Corporation in the United States and other countries. Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries. All other marks are the property of their respective owners. Corporate Headquarters: 100 Enterprise Way, Scotts Valley, CA 95066-3249 • 831-431-1000 • www.borland.com • Offices in: Australia, Brazil, Canada, China, Czech Republic, France, Germany, Hong Kong, Hungary, India, Ireland, Italy, Japan, Korea, the Netherlands, New Zealand, Russia, Singapore, Spain, Sweden, Taiwan, the United Kingdom, and the United States. • 13747